TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JUKKA SALMINEN

VIKASIETOINEN MONI-ISÄNTÄ-REPLIKOINTI JOHTAMISJÄRJESTELMÄYMPÄRISTÖSSÄ

Diplomityö

Tarkastaja: professori Hannu-Matti Järvinen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-neuvoston kokouksessa 9. huhtikuuta 2014

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
SALMINEN, JUKKA: Vikasietoinen moni-isäntä-replikointi johtamisjärjestelmäympäristössä
Diplomityö, 48 sivua
Huhtikuu 2016
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Hannu-Matti Järvinen
Avainsanat: Johtamisjärjestelmät, replikointi, optimistinen rinnakkaisuuden hallinta

Hajautettujen johtamisjärjestelmien tiedonhallintamekanismien suunnittelussa keskeinen vaatimus on, että järjestelmää käyttävän joukon toimintakyky ei vaarannu, vaikka yhteys muihin joukkoihin ei olisi jatkuvaa. Tämä on tärkeää, koska johtamisjärjestelmien toimintaympäristössä on yleistä, että hajautusverkko ei ole aina täydellisen toimintakykyinen esimerkiksi vihollisen toimista johtuen. Toisaalta tiedon jakaminen ja yhtenäisen tilannekuvan muodostaminen joukkojen kesken on tehokkaan yhteistoiminnan keskeisimpiä edellytyksiä.

Tämän työn tavoitteena oli suunnitella ja toteuttaa replikointiratkaisu, joka vastaa näihin tiedonhallintaan kohdistuviin haasteisiin. Ratkaisu toteutettiin osaksi Insta DefSec Oy:n kehittämää laajempaa yleiskäyttöistä ohjelmistoalustaa. Työssä määriteltiin ensin mitä toiminnallisia vaatimuksia replikointiratkaisuun kohdistuu. Tämän jälkeen käytiin lävitse replikoinnin laajaa teoreettista taustaa ja koottiin yhteen erilaisia replikointiratkaisun suunnittelussa hyödyllisiä käsitteitä, suunnitteluperiaatteita ja tekniikoita.

Replikoinnin toteutustavaksi valittiin optimistinen moni-isäntä-ratkaisu. Moni-isäntä-replikointi täyttää hyvin toimintaympäristön asettamat vaatimukset, mutta tuo mukanaan omat tekniset haasteensa, kuten rinnakkaisten päivitysoperaatioiden välisten konfliktien havaitsemisen. Näihin rinnakkaisuuden hallinnan ongelmiin kiinnitettiin työssä erityistä huomiota. Lopputuloksena saatiin suunniteltua ja kehitettyä käytännön tasolla pääpiirteissään toimiva replikointiratkaisu. Ongelma-alueen haastavuudesta johtuen suunniteltu ratkaisu omaa kuitenkin myös heikkouksia, joita eroteltiin työn lopuksi. Lisäksi esiteltiin vaihtoehtoinen arkkitehtuuri, joka ratkaisee osan näistä ongelmista.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
SALMINEN, JUKKA: Fault tolerant multi-master-replication in a command and control system environment
Master of Science Thesis, 48 pages
April 2016
Major: Software Engineering
Examiner: Professor Hannu-Matti Järvinen
Keywords: Command and control systems, replication, optimistic concurrency control

A key requirement of data management in distributed command and control systems is that the troops using the system do not lose their ability to function even if network connectivity to other troops isn't continuous. This is important as command and control systems function in an environment where the distribution network is not always fully functional for example due to enemy activity. On the other hand distributing information and forming a consistent situational picture between troops is a key requirement of effective cooperation.

The goal of this thesis was to design and implement a data replication solution, which meets these data management requirements. The solution was implemented as a part of a larger general purpose software platform developed by Insta DefSec Oy. First the functional requirements for the replication solution were identified. Then a study of the broad theoretical background related to replication was made collecting together various concepts, design principles and techniques useful in the design of the solution.

Optimistic multi-master replication was chosen as the overall design of the replication solution. Multi-master replication meets the requirements of the operational environment well, but brings with it its own challenges such as the detection of conflicts between concurrent update operations. These challenges related to the concurrency were paid special attention in this thesis. As a result, a functional replication solution was designed and implemented. Due to the challenging nature of the problem at hand the solution does contain some weaknesses though, which were discussed at the end of the thesis. Additionally an alternative architecture was described addressing some of these weaknesses.

ALKUSANAT

Tämä työ on tehty työskennellessäni Insta DefSec Oy:n palveluksessa. Aloitin Instan palveluksessa kymmenen vuotta sitten viittä vaille valmiina diplomi-insinöörinä ja pienen viivästyksen päätteeksi työ on nyt vihdoin saatu päätökseen.

Haluan kiittää kollegoitani Said Benamaria ja Pertti Hokkasta tarpeeseen tulleesta rohkaisusta kirjoitustyön aloittamiseksi. Kiitokset Saidille myös pitkäjänteisestä työn ohjaamisesta. Lisäksi kiitokset kuuluvat työn tarkastajalle professori Hannu-Matti Järviselle.

Vanhempiani haluan kiittää kärsivällisyydestä ja tuesta, jota ilman tähän pisteeseen pääseminen ei olisi ollut mahdollista. Kiitokset myös kaikille ystäville ja sukulaisille virkistävästä seuranpidosta, joka on edistänyt työn valmistumista suuresti.

Tampereella 3.4.2016,

_____

Jukka Salminen

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

CAP-teoreema	Hajautettujen järjestelmien eheyden, saatavuuden ja osituksen sietokyvyn suhdetta kuvaava teoreema.
FLP-mahdottomuus-todistus	Todistus hajautettujen prosessien yksimielisyyteen pääsyn mahdottomuudesta tietyissä olosuhteissa.
Hajautusverkko	Tietoverkko, jossa sama tietosisältö pyritään välittämään kaikille verkon solmuille.
Konflikti	Samaan tietosisältöön tehtyjen päivitysten välinen ristiriita.
Solmu	Yksittäinen hajautusverkossa toimiva palvelin, joka ylläpitää tietosisältökopiota sekä vastaanottaa ja lähettää tietoa replikointia hyödyntäen.
Tilannekuva	Taistelukentän tilasta muodostettu tilannetietoisuus.

1 JOHDANTO

Hajautettujen johtamisjärjestelmien suunnittelussa keskeinen vaatimus on rakentaa tiedonhallintamekanismit niin, että järjestelmää käyttävän joukon toimintakyky ei vaarannu vaikka muiden hajautusverkkoon liittyneiden toimijoiden tuottaman tiedon saatavuus olisi ajoittain heikkoa tai jopa olematonta. Johtamisjärjestelmien kohdalla tällainen tilanne on tavanomainen sillä yksi jopa käyttötapauksissa esiintyvä toimija on vihollinen, joka pyrkii jatkuvasti haittamaan hajautusverkon toimintaa ja hajottamaan sen eri osia. Joukkojen itsenäisen toiminnan mahdollistaminen tiedon saatavuuden ollessa heikkoa on siis otettava yhdeksi suunnittelun lähtökohdista.

Toisaalta ehkä keskeisin motivaatio hajautettujen johtamisjärjestelmien rakentamiselle on yhtenäisen tilannetietouden jakaminen eri järjestelmään kytkeytyneiden joukkojen kesken mahdollistaen tiiviin ja suunnitelmallisen yhteistoiminnan. Päätöksenteossa tulee ottaa laaja-alaisesti huomioon erilaisten hajautettujen toimijoiden tuottamaa tietoa sekä välittää päätöksiin liittyvää tietoa hajautusverkkoa hyödyntäen.

Tiedonhallinnan näkökulmasta tämän kaltainen itsenäisen toiminnan ja yhteistoiminnan yhdenvertaisena näkevä järjestelmä vaatii tavanomaista enemmän suunnittelutyötä. Toiminnallisuuden toteuttaminen edellyttää käytännössä sitä, että kullakin toimijalla on oma kopio toimintansa kannalta oleellisesta tietosisällöstä, jota voidaan käyttää myös silloin, kun hajautusverkon toiminta on häiriintynyt. Toisaalta hajautusverkon ollessa toimintakuntoinen, täytyy tietosisältöön tehtyjen muokkausten välittyä myös muille toimijoille. Erityisesti yhteyskatkojen aikana tietosisältöön tehtyjen muokkausten tulee välittyä muille toimijoille yhteyksien palatessa siten, että kaikilla on lopulta yhtenäinen kuva järjestelmän käsittelemästä tietosisällöstä.

Tässä työssä tutkitaan ja suunnitellaan edellä kuvatun kaltaisen tiedonhallinnan toteuttamista tiedon replikoinnin avulla. Replikoinnin ajatuksena on ylläpitää useita erillisiä ja itsenäisesti käytettävissä olevia kopioita järjestelmän tietosisällöstä. Aihealuetta on käsitelty tieteellisessä tutkimuksessa ennenkin, joten työn tavoitteena on ensin selvittää millaisia ratkaisumalleja ongelmaan on jo olemassa, ja mitä niistä voitaisiin hyödyntää parhaiten hajautettujen johtamisjärjestelmien kontekstissa.

Replikointiratkaisua ollaan toteuttamassa osaksi Insta DefSec Oy:ssä rakenteilla olevaa laajempaa ohjelmistoalustaa. Tästä syystä työ ei ole ainoastaan teoreettinen vaan tavoitteena on muodostaa myös hyvä suunnittelullinen lähtökohta toteutustyölle. Pyrkimyk-

senä on rakentaa yleiskäyttöinen ja erilaisissa johtamisen toimialasovelluksissa mahdollisimman helposti käyttöön otettava replikointiratkaisu.

Luvussa kaksi esitellään johtamisjärjestelmäympäristön eri piirteitä ja käsitellään suunnitteluvaatimuksia, joita se asettaa replikointiratkaisulle. Lisäksi luvussa esitellään laajempaa alustaratkaisua, jonka osaksi replikointi toteutetaan ja käydään lävitse miten rakentuminen osaksi alustaa tulee huomioida replikoinnin toteutuksessa.

Luvussa kolme käydään lävitse tiedon replikoinnin teoreettista taustaa ja erilaisia tieteellisissä kirjallisuudessa esiintyviä vaihtoehtoisia replikoinnin toteutustapoja.

Luvussa neljä tarkastellaan replikoinnin teoreettista taustaa vasten sitä, millainen replikointiratkaisu voisi toimia parhaiten hajautettujen johtamisjärjestelmien ja toteutettavan alustaratkaisun kontekstissa. Luvussa esitellään myös replikointikomponentin suunnittelutyö ja käydään lävitse sen toteuttamia algoritmeja.

Luvussa viisi arvioidaan suunnitellun ja toteutetun replikointiratkaisun ongelmakohtia ja pohditaan mahdollisia tapoja parantaa ratkaisua.

Luvussa kuusi tehdään yhteenveto työn tuloksista ja arvioidaan kuinka hyvin suunniteltu ja toteutettu replikointiratkaisu saavutti sille asetetut vaatimukset.

2 TOIMINTAYMPÄRISTÖN ASETTAMAT VAATIMUKSET

2.1 Hajautetut johtamisjärjestelmät

Puolustusvoimien teknologiastrategiassa johtaminen määritellään strategiseksi osaamisalueeksi, jossa korostuu kyky johtaa hajautettuja joukkoja. Yhdeksi johtamista tukevaksi kriittiseksi teknologiaksi määritellään informaatioteknologia, jonka tavoitteisiin sisältyy niin sanotun verkostopuolustusjärjestelmän luominen [1, s. 3-4]. Verkostopuolustusjärjestelmä rakentuu useista toisiinsa verkotetuista järjestelmistä ja joukoista. Järjestelmillä voi olla useita eri käyttäjiä, jotka käyttävät järjestelmää erilailla ja eri tarpeisiin. [2, s. 16]

Yksi verkostopuolustusjärjestelmän osakokonaisuus ovat hajautetut johtamisjärjestelmät. Johtamisjärjestelmien avulla luodaan tilannetietoisuutta sekä suunnitellaan ja pannaan toimeen puolustuksen suorituskyvykkyyden käyttöä. Johtamisjärjestelmät mahdollistavat joukkojen hajauttamisen laajalle alueelle antaen paremman suojan viholliselta sekä mahdollistavat tilannekuvan muodostamisen ja reaaliaikaisen johtamisen koko taistelualueella. Yksi rakenteeltaan hajautuneeseen puolustusjärjestelmään kohdistuva vaatimus on, että se ei saa kokonaisuutena lamautua, vaikka sen osa tai osia lamautetaan tai tuhotaan. [3, s. 105; 4] Hajautusverkkoon kohdistuvien hyökkäysten seurauksena yleistä on myös tilanne, jossa joukot hajoavat toisistaan erillisiksi ryhmiksi eli niin sanotuiksi saarekkeiksi siten, että yhteistoiminta kuitenkin jatkuu kunkin saarekkeen sisällä. Toisaalta yksittäinen joukko voi joutua toimimaan myös täysin itsenäisesti tilanteessa, jossa se ei pysty muodostamaan yhteyttä hajautusverkkoon.

Kuva 1 Johtamisen eri tasot

Johtamisjärjestelmiä hyödynnetään puolustusvoimien johtamisrakenteessa niin valtakunnallisella, alueellisella kuin paikallisellakin tasolla. Johtaminen voi tämän lisäksi olla strategista, operatiivista tai taktista (kuva 1). Tästä monimuotoisuudesta johtuen hajautettujen johtamisjärjestelmien tietotekninen toimintaympäristö voi vaihdella paljonkin. Runkoverkkoon ja sen keskitettyihin konesaleihin kytkeytyneiden toimijoiden käytössä oleva tietojenkäsittely-ympäristö on tyypillisesti hallittu, taistelunkestävä ja turvallinen. [5, s. 34, s. 39-40]. Toisaalta tietoa saatetaan välittää myös jalkautuneille joukoille, jolloin saman johtamisjärjestelmän osakokonaisuus voi olla käytössä taistelijoiden mukana kulkevilla päätelaitteilla tiedonsiirtokapasiteetiltaan rajoitettujen radioyhteyksien takana [6].

2.2 Tiedonhallintaan kohdistuvia vaatimuksia

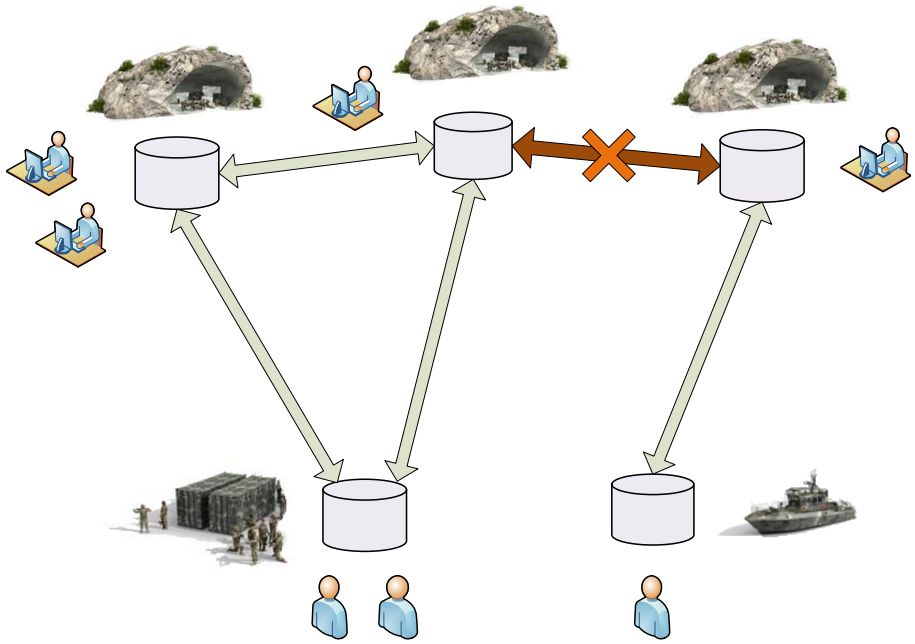
Aliluvun 2.1 kuvauksen perusteella voidaan koostaa joukko hajautettujen johtamisjärjestelmien tiedonhallintaan kohdistuvia toiminnallisia vaatimuksia.

Yhtenäisen tilannetietoisuuden, eli niin sanotun tilannekuvan, muodostaminen joukkojen kesken edellyttää, että kaikilla hajautusverkkoon kytkeytyneillä joukoilla on pääsy

samaan tietosisältöön. Kaikilla joukoilla tulee olla toiminnan niin vaatiessa myös mahdollisuus muokata tietosisältöä niin, että muokkaukset välittyvät hajautusverkon kautta muille joukoille. Tiedonsiirto hajautusverkossa tulee olla riittävän nopeaa ja suunniteltu siten, että tietoliikenneyhteyksien vaihtelevat nopeudet eivät aiheuta kohtuuttomia ongelmia järjestelmän kokonaistoiminnan kannalta. Tämä tarkoittaa käytännössä sitä, että tietoliikenneresurssien säästämiseksi tietosisällön muokkausten yhteydessä on tarkasteltava, mikä tieto on muuttunut ja välitettävä muille joukoille vain tehdyt muutokset. Lisäksi tietosisältöä tulee voida luokitella siten, että jos tietty joukko ei tarvitse toiminnassaan kaikkea hajautusverkossa saatavilla olevaa tietoa, voi se vastaanottaa verkosta vain sille tarpeellista tietoa.

Tiiviin yhteistoiminnan rinnalla on tuettava tilannetta, jossa hajautusverkon toiminta on häiriintynyt ja tiedon välittäminen verkossa ei toimi lainkaan tai toimii vain rajoitetusti. Myös käsketty radiohiljaisuus voi olla hajautusverkon toiminnan esteenä. Tällaisissa tapauksissa joukon tulee kyetä toiminaan itsenäisesti niiden tietojen pohjalta, joihin sillä on ollut pääsy hajautusverkon kautta viimeksi kun verkko on ollut toimintakuntoinen. Käytännössä joukoilla tulee olla vähintäänkin väliaikainen paikallinen tietovaranto, johon muilta toimijoilta saatua tietoa tallennetaan ja jota voidaan hyödyntää tietoliikennekatkosten aikana. Paikalliseen tietovarantoon tulee olla mahdollista tehdä muokkauksia myös yhteyskatkojen aikana ja muokattujen tietojen pitää välittyä muille joukoille yhteyksien palatessa siten, että joukoilla on lopulta yhteneväinen näkemys tilannekuvasta. Keskeiseksi ongelmaksi tilannekuvan yhtenäistämisessä muodostuu se, miten käsitellään yhteyskatkojen aikana mahdollisesti syntyneet samoihin tietoalkioihin tehtyjen muokkausten väliset konfliktit.

Hajautusverkon toiminta saattaa häiriintyä myös siten, että joukot jakautuvat erillisiksi saarekkeiksi. Kuva 2 esittää yhden mahdollisen saarekkeistumisen tavan. Tietosisällön replikoinnin tulee olla mahdollista kunkin saarekkeen sisällä. Lisäksi saarekkeiden välisten yhteyksien palatessa tulee niiden tietosisällöt pystyä yhtenäistämään siten, että sodankäyntiä voidaan jatkaa yksikäsitteisen tilannekuvan pohjalta. Myös uusien joukkojen tulee olla mahdollista liittyä hajautusverkkoon. Uuden joukon tulee liittymistoimenpiteiden jälkeen voida toimia osana hajautettua johtamisjärjestelmää yhdenvertaisena muihin joukkoihin nähden.

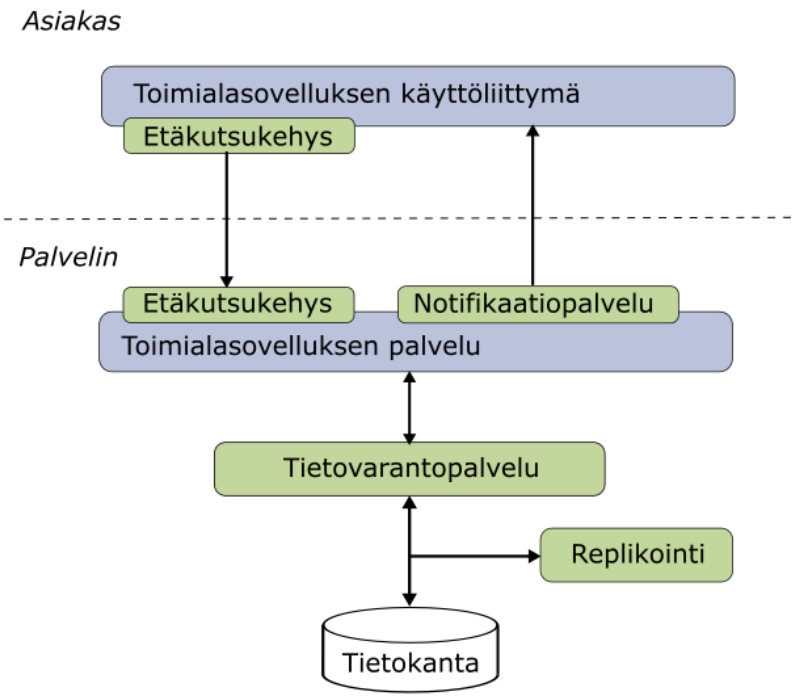


Kuva 2 Hajautusverkko osana toimintaympäristöä. Yhteyskatkot luovat verkkoon saarekkeita.

2.3 Replikointi osana alustaratkaisua

Replikointiratkaisu toteutetaan osaksi yleiskäyttöistä ohjelmistoalustaa, mikä asettaa sille tiettyjä ei-toiminnallisia vaatimuksia, mutta mahdollistaa toisaalta myös joidenkin suunnittelua helpottavien rajauksien tekemisen.

Kuva 3 esittelee alustan replikointiratkaisun kannalta keskeiset komponentit sekä kuvaa yleisellä tasolla tavan, jolla yksittäinen liiketoimintalogiikan toteuttava toimialasovellus rakentuu alustan päälle. Toimialasovellukset noudattavat arkkitehtuuriltaan perinteistä palvelukeskeistä asiakas-palvelin-arkkitehtuuria. Alusta tarjoaa erilaisia yleiskäyttöisiä komponentteja (kuvassa vihreällä), joiden päälle rakennetaan toimialalogiikan toteuttavia palveluita palvelinpäässä sekä näitä palveluita etäkutsujen avulla hyödyntäviä käyttöliittymäkomponentteja asiakaspäässä.



Kuva 3 Alustan keskeiset komponentit ja toimialasovelluksen rakenne

Alustan keskeisiä komponentteja ovat etäkutsukehys, notifikaatiopalvelu, tiedonhallintapalvelu sekä tämän työn puitteissa suunniteltava ja toteutettava replikointikomponentti. Etäkutsukehys toteuttaa synkronisen etäkutsumekanismin, jolla toimialasovelluksen käyttöliittymästä voidaan tehdä palvelinpuolen palvelulta tietoa hakevia etäkutsuja. Notifikaatiopalvelun avulla voidaan puolestaan julkaista palvelinpuolelta tietosisältöä koskevia päivityksiä käyttöliittymälle. Tietovarantopalvelu vastaa toimialasovellusten pysyvän tietosisällön hallinnoimisesta. Tietovarantopalvelu on tietokannan päälle rakentuva oliopohjaisen tiedon tallennus- ja hakurajapinnan tarjoava komponentti. Toimialasovellukset eivät ole siis suorassa yhteydessä tietokantaan vaan antavat tietosisältöolionsa tietovarantopalvelun tallennettavaksi.

Replikointikomponentti kytkeytyy osaksi tietovarantopalvelun toimintaa siten, että kaikki tietovarantopalvelun lävitse kulkevat tietosisältöolioiden lisäys-, muokkaus- ja poisto-operaatiot välitetään läpinäkyvästi myös replikoitavaksi muille hajautusverkon palvelinsolmuille. Muissa solmuissa replikointi vastaanottaa tiedonkäsittelyoperaatiot ja vie ne vastaavasti tietovarantopalveluun. Myös replikoinnin kautta vastaanotetuista tietosisällön muutoksista ilmoitetaan toimialasovelluksen palvelulle, joka välittää tiedon muutoksista solmuun kytkeytyneille käyttöliittymäsovelluksille notifikaatiopalvelua hyödyntäen. Termiä solmu käytetään jatkossa tässä työssä kuvaamaan yksittäistä verkkoon liittynyttä palvelinta, jolle tietosisältöä replikoidaan hajautusverkossa ja johon käyttöliittymäsovellukset ottavat yhteyttä.

2.4 Vaatimusten rajauksia

Replikoinnin aihealueen laajuuden takia työstä on rajattu pois joitakin vähemmän keskeisiä osakokonaisuuksia. Yksi näistä on eri tiedonsiirtotekniikoiden käsittely. Työssä oletetaan, että esimerkiksi tietosisältöön tehtyjä päivityksiä saadaan välitettyä tietosisältökopioita ylläpitävien palvelinsolmujen välillä jollakin hajautusverkossa käytössä olevalla tiedonsiirtoprotokollalla. Toinen tästä työstä pois rajattu ongelma-alue on erilaiset replikoinnin vikaantumismallit. Replikoinnin tieteellisessä tutkimuksessa on tehty paljon työtä sen selvittämiseksi, miten eri replikointiratkaisut selviävät erilaisista vikatilanteista. Esimerkiksi niin sanotussa bysanttilaisessa vikaantumismallissa (engl. byzanthine failure) tutkitaan kuinka järjestelmä selviää tilanteesta, jossa yksittäiseltä palvelimelta aletaan saada sisällöltään täysin satunnaisia tai jopa vihamielisen toimijan tarkoituksella virheellisiksi muotoilemia päivityksiä [7, s. 325]. Työ ottaa kuitenkin huomioon yleisimmät poikkeustilanteet järjestelmän toiminnassa, kuten tietoliikenneyhteyksien katkoksista johtuvat vikatilanteet.

Toinen rajaus liittyy transaktioiden käsittelyyn. Tietovarantopalvelu ei ainakaan ensimmäisessä toteutuksen vaiheessa tule tukemaan transaktioita. Tietosisältöä voidaan siis tallentaa tietokantaan tietovarantopalvelun avulla vain yksi tietoalkio kerrallaan, eikä atomisesti tietokantaan vietäviä useita tietoalkioita koskevia päivityskokonaisuuksia tueta. Tämä tarkoittaa sitä, että myöskään replikoinnin ei tarvitse välittää muihin solmuihin kuin yksittäisiä tietoalkioita koskevia päivityksiä kerrallaan. Tämä helpottaa etenkin konfliktien ratkaisemista, sillä kahden samaan tietoalkioon kohdistuneen tiedonkäsittelyoperaation välinen konflikti on huomattavasti helpommin ratkaistavissa kuin kahden mahdollisesti monimutkaisen transaktion välisen konfliktin ratkaiseminen. Tämän lisäksi toimialasovellusten oletetaan selviytyvän tilanteesta, jossa kahden tietoalkion välinen relaatio on tietovarannon tasolla hetkellisesti rikkinäinen. Tällä tarkoitetaan sitä, että jos yksittäinen tietoalkio viittaa toiseen alkioon, on toimialasovelluksen selviydyttävä tilanteesta, jossa viitattua alkiota ei välttämättä löydykään tietovarannosta. Tällainen tilanne syntyy helposti, kun esimerkiksi molemmat alkiot lisätään yhdellä kertaa tietovarantoon ja viittaava alkio tulee replikoiduksi ensimmäisenä viitatun alkion vielä odottaessa vuoroaan. Replikoinnin vastaanottavassa solmussa saadaan viittaava alkio, mutta viitattua alkiota ei ole vielä saatavilla, joten alkioiden välinen viittaus on hetkellisesti (tai oikein ajoitetun yhteyskatkon ansiosta pitempiaikaisesti) "rikkinäinen”.

3 HAJAUTETUN TIEDONHALLINNAN TEORIAA

3.1 Replikointi

Yleisellä tasolla ilmaistuna tiedon replikoinnilla tarkoitetaan tiedonhallinnan ominaisuutta, jossa järjestelmän jostakin tiedosta ylläpidetään yhtä tai useampaa kopiota. Hajautetussa tiedonhallinnassa nämä kopiot on edelleen sijoitettu tietoverkon eri palvelimille. Replikoinnilla tavoitellaan tyypillisesti parannuksia järjestelmän saatavuuteen, suorituskykyyn ja skaalautuvuuteen. Järjestelmän saatavuus paranee, kun yksittäisen tietovarannon vikaantuessa tai tietoliikenneyhteyksien katketessa voidaan siirtyä käyttämään muita tietovarannon kopioita käyttökatkokset välttäen. Suorituskykyparannuksia saavutetaan puolestaan kahdessa eri muodossa. Sijoittamalla tietovarannon kopiot maantieteellisesti eri sijainteihin lähelle tiedon käyttäjiä voidaan vähentää verkkoviiveistä johtuvia suorituskykyongelmia. Suorituskykyä saadaan parannettua myös siten, että tietovarantoihin kohdistuvat kutsut voidaan jakaa eri kopioiden välillä vähentäen yksittäiseen tietovarantoon kohdistuvaa kuormaa. Järjestelmä on tietovarantokopioiden ansiosta myös skaalautuvampi, sillä kuormituksen lisääntyessä järjestelmän suorituskykyä voidaan tukea luomalla kopioita lisää. Replikointi voi myös tietyissä tapauksissa parantaa järjestelmän luotettavuutta yksittäisen tietovarannon korruptoituessa. Jos järjestelmä ylläpitää vähintään kolmea tietovarantokopiota, voidaan kahden varannon sisältämää tietosisältöä pitää oikeana, ei-korruptoituneena, sisältönä. [7, s. 274; 8, s. 4; 9, s. 459]

Koska ohjelmistojen toiminnassa on yleistä, että tiedosta tehdään ja ylläpidetään kopioita, on hyvä selventää replikoinnin käsitteellistä eroa muihin yleisesti käytössä oleviin tiedon kopioinnin muotoihin. Välimuisti (engl. cache) on tietosisältökopio, joka sijoitetaan tietoa käyttävän asiakkaan lähelle varsinaisen tietovarannon edustalle siten, että varantoon kohdistuvaa kuormitusta saadaan kevennettyä. Välimuisteilla tavoitellaan siis tyypillisesti skaalautuvuus- ja suorituskykyparannuksia. Replikoinnista poiketen välimuistin sisältö valikoidaan useimmiten reaktiivisesti asiakkaan tekemien lukuoperaatioiden perusteella, kun taas replikointi on aktiivista ja automaattista tietosisällön kopioimista [10]. Toinen replikoinnista poikkeava tiedon kopioinnin muoto on varmuuskopiointi. Varmuuskopiointi toimii viimeisenä vikasietoisuuden ja siten saatavuuden takaajana tilanteessa, jossa pääasiallisen tietovarannon tiedot jostain syystä menetetään [11]. Replikoinnista poiketen suorituskykyä ja skaalautuvuutta varmuuskopioilla ei kuitenkaan pyritä parantamaan. Välimuistit ja varmuuskopiointi voidaan toisaalta nähdä myös

replikoinnin osittaisina toteutuksina, joista on jätetty tiettyjä ominaisuuksia pois ongelma-alueen rajaamiseksi. Samalla on luovuttu joistakin täysverisen replikointiratkaisun tarjoamista hyödyistä. Tästä rajauksesta johtuen yleiskäyttöisen replikointiratkaisun suunnittelussa tarvitsee tarkastella tietosisältökopioiden ylläpitämisen ongelmaa laaja-alaisemmin kuin mitä näissä ratkaisuissa on tehty.

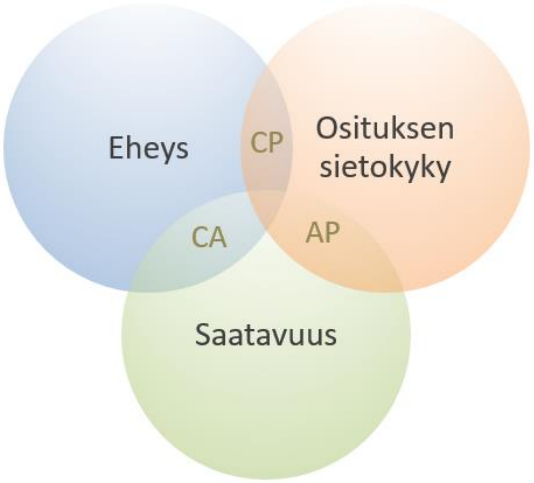
3.2 CAP-teoreema ja FLP-mahdottomuustodistus

Ideaalinen replikointiratkaisu tarjoaisi kaikki useiden hajautettujen tietovarantokopioiden ylläpitämisen tarjoamat hyödyt näyttäen kuitenkin ulospäin siltä kuin järjestelmässä olisi vain yksi ainoa tietovaranto. Kahden hajautettujen tiedonhallintaratkaisujen ominaisuuksia kuvaavan säännön, CAP-teoreeman ja FLP-mahdottomuustodistuksen, mukaan tällaisen loputtomasti skaalautuvan, aina eheän ja erittäin vikasietoisen tiedonhallinnan toteuttaminen on kuitenkin mahdotonta. Replikoinnin käytännön toteutusten tulee siis väistämättä toimia näiden teoreemien asettamien rajoitusten puitteissa.

FLP-mahdottomuustodistus, joka saa nimensä todistuksen laatijoiden Michael J. Fischerin, Nancy A. Lynchin ja Michael S. Patersonin etukirjaimista, osoittaa että usean hajautetun prosessin välillä on mahdotonta päästä kaikissa tilanteissa yksimielisyyteen (engl. consensus) niiden yhteisesti ylläpitämästä tiedosta, jos prosessien välinen kommunikaatio on asynkronista ja yksikin prosesseista voi vikaantua. Toisin sanoen jos prosessien välisen kommunikoinnin viiveille ei ole asetettu kiinteää ylärajaa, voi yhdenkin prosessin vikaantuminen johtaa siihen, että järjestelmän tietosisältöä koskevan konsensuksen muodostamista ei saada koskaan vietyä päätökseen (engl. termination) [12]. Käytännössä todistus johtaa siihen, että asynkronista kommunikointia hyödyntävien järjestelmien suunnittelussa tulee tehdä valinta tiedonkäsittelyn turvallisuuden ja järjestelmän elävyyden välillä, kun turvalliseksi tiedonkäsittelyksi määritellään kyky muodostaa yksimielinen ja validi tietosisältö prosessien kesken ja elävyys kyvyksi viedä tiedon hajauttaminen prosessien välillä päätökseen. [13]

CAP-teoreeman mukaan hajautetussa tiedonhallintaratkaisussa voidaan saavuttaa maksimissaan kaksi kolmesta seuraavista ominaisuuksista samaan aikaan: eheys (engl. consistency), saatavuus (availability) ja osituksen sietokyky (partition tolerance) (kuva 4). Eheydellä tarkoitetaan ominaisuutta, jossa järjestelmään kohdistuvat kirjoitusoperaatiot suoritetaan atomisesti kaikkiin sen tietovarantoinstansseihin aivan kuin ne suoritettaisiin, jos tietovarantoja olisi vain yksi kappale pitäen näin kaikki instanssit jatkuvasti samansisältöisinä. Toisin ilmaistuna tietyn kirjoitusoperaation jälkeen tehtyjen lukuoperaatioiden tulee palauttaa kyseisen kirjoitusoperaation tietosisältö, riippumatta siitä mihin tietovarantoinstanssiin lukuoperaatio kohdistuu [14, s. 52]. Saatavuudella tarkoitetaan ominaisuutta, jossa jokainen järjestelmän yksittäiseen tietovarantoinstanssiin kohdistettu tiedonhakuoperaatio saa ennemmin tai myöhemmin vastauksen [14, s. 53]. Saatavuuden käsite voidaan nähdä myös suorituskyvyn ja vikasietoisuuden yhdistelmänä:

järjestelmän tulee pysyä toimintakuntoisena ja vastata siihen kohdistuviin kutsuihin nopeasti myös tilanteessa, jossa osa tietovarantokopioista olisi rikkoutunut tai saavuttamattomissa [15, s. 23]. Osituksen sietokyvyllä puolestaan tarkoitetaan sitä, että järjestelmä pystyy jatkamaan toimintaansa myös tilanteessa, jossa tietovarantoinstanssien väliset tietoliikenneyhteydet eivät toimi rikkoen hajautetun järjestelmäkokonaisuuden erillisiin osiin. [14, s. 53; 15, s. 23]. Tämän työn puitteissa tällaista osittumista kutsutaan myös saarekkeisiin jakautumiseksi.



Kuva 4 CAP-teoreema visualisoituna. Kolmen ympyrän leikkausta on mahdotonta saavuttaa samassa järjestelmässä.

Saatavuus ja osituksen sietokyky ovat useimmiten luonteeltaan binäärisiä ominaisuuksia: järjestelmä joko kykenee vastaamaan siihen kohdistettuun kutsuun järkevässä ajassa tai ei, ja järjestelmä joko pysyy toimintakuntoisena tietovarantojen välisten yhteyksien katketessa tai ei. Eheys on ominaisuutena kuitenkin hienojakoisempi ja vaikuttaa replikointiratkaisun suunnitteluun laajemmin. Tästä syystä eri eheyden malleja käsitellään tarkemmin aliluvussa 3.3.

3.3 Eheyden eri malleja

Eheydellä tarkoitetaan täydellisessä muodossaan ominaisuutta, jossa järjestelmä näyttää ulospäin siltä kuin sillä olisi vain yksi aina ajan tasalla oleva tietovaranto [16, s. 23]. Replikointia hyödyntävässä täydellisen eheyden omaavassa hajautetussa järjestelmässä tietosisällön kopioiden tulee siis ainakin ulospäin näyttää olevan toisiinsa nähden jatkuvasti sisällöltään identtisiä [17, s. 7]. Tällaista eheyden muotoa kutsutaan myös vahvaksi (engl. strong), atomiseksi (atomic), linearisoituvaksi (linearizable) tai yhden kopion (single copy) eheydeksi [17, s. 4; 14, s. 52; 18, s. 43]. Vahvan eheyden omaavien järjestelmien etuna on se, että sovelluskehittäjien ei tarvitse välittää siitä, että tiedonhallinnan osana tehdään replikointia, koska tietovaranto käyttäytyy yksittäisen tietovarantokopion tavoin. Huonona puolena vahvan eheyden ylläpitäminen vaatii usein synkronista kommunikointia eri hajautettujen tietovarantokopioiden välillä, mikä etenkin kopioiden

määrän kasvaessa voi heikentää järjestelmän suorituskykyä [7, s. 276]. Ongelmallinen on myös tilanne, jossa verkkoyhteydet eivät toimi, mutta vahva eheys kopioiden välillä tulisi kuitenkin säilyttää johtaen CAP-teoreeman mukaisesti järjestelmän saatavuuden heikentymiseen.

Luovuttaessa vahvasta eheydestä voidaan replikoinnin suunnittelussa hyödyntää koko joukkoa erilaisia eheyden malleja riippuen siitä, millaisia poikkeamia eri kirjoitus- ja lukuoperaatioiden voidaan sallia tietosisällössä näkevän, kun operaatiot kohdistetaan eri kopioihin eri ajan hetkillä [19]. Järjestelmä voi esimerkiksi noudattaa niin sanottua sekventiaalisen eheyden mallia (engl. sequential consistency), jossa kaikki eri tietokopioita käsittelevät prosessit näkevät tekemiensä lukuoperaatioiden kautta kaikki kirjoitusoperaatiot samassa järjestyksessä kuitenkin ilman vaatimusta siitä, että järjestyksen tulisi olla sama kuin alkuperäinen kirjoitusoperaatioiden globaali suoritusaikajärjestys, mikä kuuluu vahvan eheyden vaatimuksiin [7, s. 282]. Kausaalisessa eheydessä (engl. causal consistency) puolestaan edellytetään vain sitä, että ne kirjoitusoperaatiot, joilla voi olla jokin samaan tietoon kohdistuneiden luku- ja kirjoitusoperaatioiden perusteella päätel-tävissä oleva syy-seuraussuhde, näkyvät muille prosesseille aina samassa järjestyksessä. Muut kirjoitusoperaatiot voivat näkyä missä järjestyksessä tahansa [7, s. 284].

Yleisesti ottaen kaikkia sekventiaalista eheyttä heikompia eheyden malleja, joissa eri tietosisältökopioiden välinen ajoittainen epäeheys sallitaan, kutsutaan heikoksi eheydeksi (engl. weak consistency) [20]. Sallimalla heikko eheys saavutetaan järjestelmässä tyypillisesti parannuksia suorituskykyyn, kun kirjoitusoperaatioita ei tarvitse välittää synkronisesti kaikkiin tietosisältökopioihin. Heikon eheyden huonona puolena on vastaavasti se, että sovelluskehittäjien tulee tietää millaisia puutteita tiedonhallinnan eheydessä on ja ottaa ne huomioon omassa sovelluslogiikassaan [17, s. 10]. Yksi heikkoa eheyttä kuvaava termi on niin sanottu ajan myötä eheä -malli (engl. eventual consistency), jolla viitataan siihen, että järjestelmä voi olla pitkiäkin aikoja epäeheässä tilassa, mutta lopulta eheys kuitenkin saavutetaan. Tämä malli mahdollistaa järjestelmän saatavuuden myös verkon ollessa osittunut verkkoyhteyksien puuttumisen takia. Ajan myötä eheä -mallin mukainen eheys saavutetaan tyypillisesti niin, että asiakkaat suorittavat kirjoitusoperaatioita tietosisältökopioihin ja operaatiot välitetään yhteyksien toimiessa asynkronisesti muihin kopioihin taustaprosessina. Eri kopioihin tehtävien päivitysten samanaikainen suorittaminen voi johtaa konfliktissa oleviin kirjoitusoperaatioihin, joten tässä mallissa on myös välttämätöntä toteuttaa jokin konfliktinratkaisumekanismi [17, s. 12].

3.4 Järjestys ja aika

Järjestelmän tietosisältöön kohdistuvien kirjoitusoperaatioiden voidaan ajatella vievän järjestelmää tilasta toiseen operaatioiden suoritusjärjestyksen määrittelemän sekvenssin mukaisesti. Hajautetussa tiedonhallinnassa tämän suoritusjärjestyksen määrittäminen

muodostuu keskeiseksi ongelmaksi. Vahvan eheyden ylläpitämisen edellytyksenä on tyypillisesti se, että tiedonkäsittelyoperaatiot pystytään laittamaan jonkinlaiseen loogiseen järjestykseen ja operaatiot saadaan vietyä kaikkiin tietosisältökopioihin samassa järjestyksessä [17, s. 10]. Jos taas sallitaan ajoittainen tiedon epäeheys, tarvitaan mekanismi jonka avulla pystytään havaitsemaan operaatioiden rinnakkaisuus (engl. concurrency) ja ratkaisemaan operaatioiden välillä oleva konflikti. Tässä luvussa esitellään joitakin hajautetun tiedonhallinnan yhteydessä yleisesti käytössä olevia tiedonkäsittelyoperaatioiden järjestämisen menetelmiä.

Tiedonkäsittelyoperaatioiden järjestyksestä puhuttaessa on mielekästä hyödyntää kahta matemaattista määritelmää: osittaista järjestystä ja täydellistä järjestystä. Osittaisessa järjestyksessä kaikille joukon elementeille a, b ja c pätee binäärinen järjestystä kuvaava relaatio ”≤”, joka on

- Refleksiivinen: $a \leq a$
- Antisymmetrinen: Jos $a \leq b$ ja $b \leq a$, niin $a = b$
- Transitiivinen: Jos $a \leq b$ ja $b \leq c$, niin $a \leq c$.

Täydellinen järjestys edellyttää näiden vaatimusten lisäksi totaalisuutta: Joko $a \leq b$, tai $b \leq a$. Osittaisen järjestyksen omaavassa tiedonkäsittelyoperaatioiden joukossa osalla operaatioista on siis toisiinsa nähden määritelty järjestys, mutta ei kaikilla. Täydellisessä järjestyksessä kaikilla operaatioilla on keskinäinen järjestys. [21; 22]

Ehkä intuitiivisin järjestyksen määrittämiseksi käytettävissä oleva väline on aika. Jos kaikille järjestelmän kirjoitusoperaatioille pystytään määrittämään globaalisti pätevä aikaleima, on operaatioilla triviaalisti määriteltävä täydellinen järjestys. Aikaleimojen käyttämisen edellytykseksi muodostuu se, miten eri hajautetun järjestelmän solmujen kellot saadaan synkronoitua riittävän tarkasti. Kellojen synkronointi on itsessään laajalti tutkittu aihealue ja ongelman ratkaisemiseksi on kehitelty myös yleisesti käytössä olevia protokollia ja algoritmeja, kuten NTP (Network Time Protocol), Cristianin algoritmi ja Berkeley-algoritmi [23; 24; 25]. Näiden algoritmien käyttö rajoittuu kuitenkin ympäristöihin, joissa verkkoyhteydet palvelimien välillä ovat hyvät. Nykyisin myös GPS toimii yhtenä globaalisti saatavilla olevan tarkan kellonajan lähteenä, mutta vaatii erityistä laitteistoa GPS-signaalin vastaanottamiseksi [26].

Aikaleimojen avulla ei kuitenkaan välttämättä pystytä päättelemään operaatioiden välistä mahdollista kausaliteettia; siis sitä onko yhden tiedonkäsittelyoperaation suorittaminen vaikuttanut toiseen. Kausaliteetin kuvaamisessa voidaan hyödyntää niin sanottua aikaisempi tapahtuma -suhdetta (engl. happened before), joka määrittelee operaatioille osittaisen järjestyksen hajautetussa järjestelmässä. Aikaisempi tapahtuma -suhde ”→” määritellään seuraavasti.

- Jos samassa hajautetun järjestelmän prosessissa/solmussa suoritettu operaatio/tapahtuma a on suoritettu ennen operaatiota b, pätee a→b.
- Jos a kuvaa yhden prosessin lähettämää viestiä ja b saman viestin vastaanottamista toisessa prosessissa, niin a→b
- Jos a→b ja b→c, niin a→c.

Jos a↛b ja b↛a, sanotaan että a ja b ovat rinnakkaisia. Suhteen a→b voidaan nähdä kuvaavan operaatioiden välistä kausaliteettia siten, että a on voinut vaikuttaa b:hen. Rinnakkaiset operaatiot eivät voi vaikuttaa toisiinsa kausaalisesti. [27, s. 559]

Tietoa aikaisempi tapahtuma -suhteen olemassaolosta voidaan ylläpitää käytännössä niin sanottujen loogisten kellojen avulla. Lamportin kello on yksinkertainen fyysisestä kellosta riippumaton laskuri, joka liittää yksittäisiin operaatioihin numeroarvon. Se voidaan määritellä funktiona C, joka määrittää jokaiselle prosessissa i suoritettavalle operaatiolle a kellon arvon $C_i(a)$. Jotta kellon arvoista voitaisiin tehdä järjestykseen liittyviä päättelyjä, määritellään kaikille operaatioille a ja b ehto: jos a→b, niin $C(a) < C(b)$. Tämä ehto täyttyy silloin kun hajautetut prosessit toimivat seuraavien sääntöjen mukaan:

1. Jokainen prosessi P_i kasvattaa omaa loogista kelloaan C_i ennen jokaista operaatiota.
2. Jokaisen prosessin P_i lähettämän viestin m yhteydessä välitetään toisille prosesseille kellon sen hetkinen arvo T_m.
3. Vastaanottaessaan viestin m, prosessi P_j asettaa oman kellonsa arvon C_j arvoon $\max(C_j, T_m)$.

Huomattavaa on, että vaikka $C(a) < C(b)$, niin ei voida päätellä toisinpäin, että a→b. Yksittäisen prosessin paikallisten operaatioiden kohdalla näin voitaisiin päätellä, mutta usean prosessin mahdollisesti rinnakkaisten operaatioiden kohdalla ei. Näin ollen pelkän Lamportin kellon avulla ei voida tehdä päätelmiä kaikkien järjestelmän operaatioiden välisestä kausaliteetista. [27, s. 560]

Lamportin kellon voidaankin ajatella olevan lähtökohta niin sanotulle vektorikelloalgoritmille (engl. vector clock), jonka avulla kausaliteetti, tai mahdollinen rinnakkaisuus, pystytään päättelemään kaikkien operaatioiden välillä. Vektorikellossa yhden kokonaislukulaskurin sijasta ylläpidetään kullekin operaatiolle taulukkoa $[C_i, C_j \ldots C_n]$, joka sisältää oman laskurin jokaiselle järjestelmän hajautetulle prosessille. Näitä laskurien arvoja ylläpidetään seuraavien sääntöjen mukaisesti:

1. Alussa kaikki arvot ovat nollia.

2. Paikallista (prosessin omaa) kelloa päivitetään vähintään kerran ennen jokaista operaatioita.
3. Koko laskuritaulukko sen hetkisine arvoineen välitetään jokaisen lähetetyn viestin mukana muille prosesseille.
4. Vastaanotettaessa viestiä muodostaa vastaanottava prosessi laskuritaulukolle uudet arvot valitsemalla aiemmasta ja vastaanotetusta taulukosta maksimiarvot kullekin prosessille. Tämän lisäksi vastaanottava prosessi kasvattaa omaa laskuritaulukon arvoaan yhdellä.
5. Laskurien arvoja ei koskaan pienennetä. [28, s. 58]

Jos operaatioiden a ja b laskuritaulukoille pätee kaikkien laskuritaulukon arvojen osalta pareittain $C(a) \leq C(b)$ ja vähintään yhdelle arvoparille $C(a) < C(b)$, voidaan nyt päätellä, että a→b. Näin muodostetun loogisen kellon avulla pystytään havaitsemaan kausaliteetti tai sen puuttuminen minkä tahansa kahden järjestelmässä suoritetun operaation välillä. Käytännössä mekanismin hyödyllisyys tulee esiin etenkin silloin, kun halutaan tietää, onko samaan tietoalkioon tehty rinnakkaisia, konfliktissa olevia, muutoksia kahdessa eri järjestelmän prosessissa. Otettaessa viestiä vastaan toiselta prosessilta, voidaan tarkastella vastaanotetun ja paikallisen prosessin tiedossa olevan viimeisimmän operaation osalta päteekö a↛b ja b↛a. Jos pätee, on havaittu konflikti.

3.5 Replikointiratkaisuja jaottelevia piirteitä

Replikoinnin useita vuosikymmeniä kattava tutkimushistoria pitää sisällään lukuisia erilaisia lähestymistapoja replikoinnin toteuttamiselle, jotka ovat syntyneet, kun ratkaisulta vaadittavia ominaisuuksia on painotettu erilailla. Replikoinnin sinällään yhtenäisen perusajatuksen, tietosisältökopioiden ylläpitämisen, selkeydestä huolimatta, mitään yhtenäistä toteutustapaa ei siis ole olemassa. Valitettavasti monille replikointiratkaisujen ominaisuuksia kuvaaville termeille on myös kehittynyt monia melkein, mutta ei aivan, samaa tarkoittavia synonyymejä. Tässä luvussa esitetään tiivistetysti keskeisimmät erilaiset replikoinnin toteuttamisen lähestymistavat ja termit, joilla näitä kuvataan.

Replikoinnin toteutustapoja voidaan jaotella korkealla tasolla seuraavien piirteiden perusteella [9, s. 460].

- Tietovarantokopioiden erilaiset roolit asiakkaan näkökulmasta. Replikointiratkaisut voidaan jaotella sen mukaan nähdäänkö yksi tietosisältökopio kirjoitusoperaatioiden suhteen ensisijaisena muihin verrattuna vai ovatko kaikki kopiot toisiinsa nähden yhdenvertaisia. Tämän työn puitteissa näistä kahdesta eri lähestymistavasta käytetään termejä isäntä–renki- (yhden kopion ensisijaisuus kirjoitusoperaatioiden suhteen) ja moni-isäntä-replikointi (kopioiden yhdenvertaisuus). Isäntä–renki-replikoinnista käytetään englanninkielisiä termejä master-slave, primary–backup, primary–copy ja active–passive. Moni-isäntä-termin

vastineita ovat multi-master, multi-primary, master-master, update anywhere ja active–active. Käsitteet passive- ja active-replikointi kuvataan myös usein hyvin samantapaisesti kuin isäntä–renki- ja moni-isäntä-replikointi, mutta näiden merkityksestä on myös jonkin verran erilaisia käsityksiä [29; 30; 17 s. 20]. Nämä toteutustavat on kuvattu tarkemmin aliluvussa 3.5.1

- Tietosisällön päivitysten välittäminen tietovarantokopioiden välillä. Replikointiratkaisut voidaan jakaa synkronisiin ja asynkronisiin ratkaisuihin riippuen siitä välitetäänkö tietosisältöön tehdyt muutokset kaikkiin kopioihin osana samaa kirjoitusoperaatiota siten, että kirjoitusoperaatiota suorittava asiakas odottaa vai erillisenä taustaoperaationa varsinaisen kirjoitusoperaation jälkeen. Synkronista replikointia kuvataan usein englanninkielisellä termillä eager ja asynkronista termillä lazy.
- Tietovarantokopioiden eheyden ylläpitäminen. Tämän piirteen osalta replikointiratkaisut voidaan jakaa niin sanottuihin pessimistisiin ja optimistisiin ratkaisuihin [18, s. 43]. Pessimistisyydellä tarkoitetaan toteutustapaa, jossa eri tietovarantokopiot pyritään pitämään aina samansisältöisinä. Optimistisissa ratkaisuissa puolestaan eri kopioiden sisältöjen sallitaan olosuhteiden niin vaatiessa erkaantua toisistaan sillä seurauksella, että kahden kopion välille saattaa tulla sisällöllisiä ristiriitoja eli niin sanottuja konflikteja, jotka pitää erikseen ratkaista. Käytännössä eheyden ylläpitämisen toteutustapa kulkee käsi kädessä päivitysten välittämisen synkronisuuden kanssa; pessimistiset ratkaisut vaativat synkronista tiedonsiirtoa, kun taas optimistiset ratkaisut voivat olla asynkronisia. Pessimistinen ja optimistinen replikointi on kuvattu tarkemmin aliluvuissa 3.5.2 ja 3.5.3.

3.5.1 Isäntä–renki- ja moni-isäntä-replikointi

Yksi tapa jaotella erilaisia replikointiratkaisuja pohjautuu kysymykseen siitä onko eri tietosisältökopioilla samanlaiset roolit järjestelmän kokonaistoiminnassa. Niin sanotussa isäntä–renki-replikoinnissa kaikki asiakkailta tulevat kirjoitusoperaatiot kohdistetaan yhteen ensisijaiseen tietovarantoon [15, s. 208]. Tämän varannon rinnalla on yksi tai useampia toissijaisia tietovarantokopioita, joihin tietoa kopioidaan joko asynkronisesti esimerkiksi kirjoitusoperaatiot koostavaa transaktiolokia hyödyntäen tai synkronisesti esimerkiksi kaksivaiheisen kommittointiprotokollan (engl. two phase commit, 2PC) avulla [32; 9, s. 465]. Sallimalla kirjoitusoperaatiot vain yhteen tietovarantoon vältytään kopioiden välisiltä epäeheysongelmilta, sillä eri kopioihin ei voi kohdistua samanaikaisia konfliktin aiheuttavia kirjoitusoperaatioita. Toissijaiset tietovarantokopiot toimivat kahdessa roolissa. Ne pitävät yllä tietosisällön varmuuskopioita ja voivat ensisijaisen tietovarannon vikaantuessa ottaa sen roolin itsellensä varmistaen näin järjestelmän saatavuuden. Tietovarantoihin kohdistuvat lukuoperaatiot voidaan puolestaan jakaa kaikkien kopioiden kesken, jolloin järjestelmän suorituskyky paranee [33]. Moni-isäntä-replikoinnin voidaan näin ollen ajatella muistuttavan monin tavoin sekä järjestelmän saatavuutta parantavaa varmuuskopiointia, että lukuoperaatioita nopeuttavaa välimuisti-

ratkaisua toissijaisten, vain luettavaksi tarkoitettujen, kopioiden muodossa. Isäntä–renki-replikoinnin haasteena on usein niin sanotun split brain -ongelman välttäminen, jossa esimerkiksi tietoliikenneyhteyksien vikaantumisesta aiheutuvan kommunikaatiokatkoksen johdosta menetetään yksikäsitteinen ymmärrys siitä, mikä tietovarantokopio on ensisijainen [15, s. 40]. Tällöin kaksi tietovarantokopiota saattaa ottaa itsellensä samaan aikaan ensisijaisen varannon tehtävät sekoittaen järjestelmän toiminnan.

Moni-isäntä-replikoinnissa kaikki tietovarannon kopiot toimivat yhteneväisessä roolissa ja voivat vastaanottaa samanaikaisesti sekä luku-, että kirjoitusoperaatioita [9, s. 465, s. 467]. Tällöin saavutetaan samankaltaiset saatavuus- ja suorituskykyhyödyt kuin isäntä–renki-replikoinnissa sillä lisäyksellä, että myös kirjoitusoperaatiot voidaan jakaa tietovarantokopioiden välillä. Myös moni-isäntä-replikointi voidaan toteuttaa sekä synkronisena tai asynkronisena. Keskeisenä heikkona puolena isäntä–renki-replikointiin verrattuna moni-isäntä-replikoinnissa on mahdollista, että eri tietosisältökopioihin yritetään tehdä samanaikaisesti kahta konfliktin aiheuttavaa kirjoitusoperaatiota, jolloin kopioiden välinen eheys vaarantuu. Replikointiratkaisun täytyy tällöin joko estää samanaikaiset kirjoitusoperaatiot, mikä vaatii käytännössä synkronista kirjoitusoperaatioiden välittämistä tietovarantokopioiden välillä, tai sallia samanaikainen suoritus ja kyetä ratkaisemaan syntynyt konflikti asynkronisen kirjoitusoperaatioiden välittämisen yhteydessä. Koska moni-isäntä-replikoinnissa eri tietovarantokopiot toimivat samalla tavalla tarjoten tietosisältöön luku- ja kirjoitusoikeudet, on mahdollista toteuttaa järjestelmä, jossa kukin tietovaranto on käytettävissä itsenäisesti muista kopioista riippumatta. Tällainen replikointiratkaisu soveltuu erityisen hyvin käyttötarkoitukseen, jossa verkkoyhteyksien olemassaolo ei ole itsestäänselvyys ja tietovarantojen käyttäjien itsenäinen toiminta on osa järjestelmän tavanomaista toimintaa [18, s. 43]. Järjestelmän saatavuus voidaan siis taata myös monissa tilanteissa, joissa isäntä–renki-ratkaisu lakkaisi toimimasta.

3.5.2 Pessimistinen replikointi

Sinällään termi “pessimistinen” ei esiinny kovinkaan usein seuraavien replikointiratkaisujen kuvausten yhteydessä, mutta se on kuitenkin terminä hyödyllinen tiettyjen historiallisesti merkittävien replikoinnin toteutustapojen kuvaamisessa. Pessimististen replikointiratkaisujen keskeisenä piirteenä on pyrkimys luoda illuusio yhdestä korkean tason saatavuuden ja vahvan eheyden omaavasta tietovarantokopiosta. Erityisesti pyrkimys jatkuvan vahvan eheyden ylläpitämiseen kopioiden välillä on keskeinen pessimistisen replikoinnin tavoite. Tämä tavoite voidaan saavuttaa useilla eri tavoilla, mutta perinteisten pessimististen ratkaisujen yhteisenä nimittäjänä on se, että pääsy yksittäiseen kopioon estetään, ellei se ole todistettavasti eheässä tilassa, mikä voi heikentää järjestelmän saatavuutta. Järjestelmässä voi siis olla yksi tai useampia kopioita, joiden katsotaan omaavan järjestelmän oikean tilan, muiden ollessa epäeheässä tilassa ja suljettuna normaalin toiminnan ulkopuolelle. Pessimistisen replikoinnin etuna on vahvan eheyden tuoma yksinkertaisuus; tietovarantoa käyttävän ohjelmiston ei tarvitse kyetä havaitsemaan, eikä ratkaisemaan tiedonkäsittelyn rinnakkaisuudesta aiheutuvia konflikteja.

Heikkona puolena pessimistinen replikointi puolestaan kärsii saatavuuden heikkenemisestä ja suorituskyvyn laskusta mikäli tietovarantokopioiden määrä kasvaa suureksi tai tietoliikenneyhteyksien laadusta ei ole takeita. [18, s. 43]

Keskeisessä roolissa vahvasti eheiden kopioiden ylläpitämisessä ovat niin sanotut konsensusprotokollat tai -algoritmit. Näistä käytetään myös termiä atominen kommittointiprotokolla. Esimerkiksi kaksivaiheisen kommittointiprotokollan (engl. two phase commit, 2PC) avulla voidaan tiedon päivittäminen suorittaa atomisesti useaan verkkoon hajautettuun kopioon varmistuen siitä, että päivitys tehdään kaikkiin kopioihin samanlaisena tai ei ollenkaan. Ajatuksena on tehdä päivitys kahdessa vaiheessa siten, että transaktio valmistellaan ensin kussakin kopiossa suoritettavaksi ja kun kaikki kopiot ilmoittavat olevansa valmiita, kommittoidaan päivitys kopioihin yhdellä kertaa [15, s. 326]. Yksi solmu toimii tällöin transaktion suorituksen koordinoijana. Kolmivaiheinen kommittointiprotokolla (3PC) lisää protokollaan yhden vaiheen lisää, joka parantaa protokollan toimintavarmuutta tilanteessa, jossa transaktiota koordinoiva solmu vikaantuu kesken transaktion suorituksen. Sinfonia on toinen 2PC:n perusajatusta jatkava hajautettujen transaktioiden suorittamisen menetelmä, jossa kommittointiprotokolla rakentuu niin sanottujen minitransaktioiden varaan. [15, s. 333; 34; 35].

Erityisen merkittävä ja laajalti hyödynnetty konsensusprotokolla on nimeltään Paxos. Paxosin toimintaperiaate muistuttaa tietyissä määrin 2PC-algoritmia, jonka on itse asiassa osoitettu olevan Paxos-algoritmin rajoitettu muoto [36, s. 134]. Paxosissa yksi solmu ottaa uusien arvojen ehdottajan roolin (engl. proposer) muiden toimiessa hyväksyjinä (engl. acceptor). Ehdottaja tekee uusia arvoja valmistelevia, täydellisen järjestyksen omaavia, kutsuja hyväksyjille. Jos ehdottaja saa vastauksen hyväksyjien enemmistöltä, lähettää se hyväksyjille hyväksyntäpyynnön, jolloin arvo astuu voimaan [37]. Tiettyjen uusien arvojen ehdottamiseen liittyvien lisäpiirteiden ansiosta lopulta kaikki solmut päätyvät konsensukseen uudesta arvosta. Algoritmin oikeellisuus ei kärsi verkon häiriötilanteista, eikä esimerkiksi tilanteesta, jossa useampi solmu yrittää toimia ehdottajana samaan aikaan. Paxos-algoritmi kuvaa perusmuodossaan tavan, jolla yksittäisestä uudesta arvosta päästään konsensukseen solmujen kesken. Multi-Paxos puolestaan on algoritmin laajennos, joka kuvaa useiden arvojen peräjälkeisen asettamisen tehden siitä käyttökelpoisemman todellisissa järjestelmissä [38, s. 400]. Toimivuudestaan huolimatta Paxosia on pidetty monimutkaisena ja vaikeasti ymmärrettävänä algoritmina [37]. Raft-konsensusalgoritmi on Paxosin modernisoitu ja yksinkertaistettu versio, joka tarjoaa samankaltaiset vikasietoisuus- ja suorituskykyominaisuudet, mutta muodossa, joka pyritty suunnittelemaan helpommin ymmärrettäväksi [39].

Näitä ja muita konsensusprotokollia voidaan käyttää hyödyksi niin sanotun tilakonereplikoinnin (engl. state machine replication) toteuttamisessa. Tilakonereplikoinnissa järjestelmän solmut muodostavat konsensuksen kustakin järjestelmässä seuraavaksi suoritettavasta komennosta ja vievät komentoja suorittaessaan solmukokonaisuutta tilasta

toiseen tilakoneen tavoin. Paxos- ja Raft-pohjaisissa ratkaisuissa yksi solmuista toimii tyypillisesti johtajana ottaen vastaan asiakkailta tulevia komentoja ja toimien ehdottajan roolissa. Komennot voidaan kerätä esimerkiksi konsensusprotokollan avulla hajautettuun komentolokiin, joka ylläpitää listaa kaikista suoritettavista komennoista täydellisessä järjestyksessä. Näin toteutettu replikointi toimii oikein, on vikasietoisen, sietää verkkohäiriöitä (kunhan enemmistö solmuista on saavutettavissa) ja on normaalitapauksessa kohtuullisen suorituskykyinen. [37; 39, s. 2].

Kaiken kaikkiaan konsensusprotokollien ja pessimistiksi luettavien replikointialgoritmien historia on pitkä ja varsin vaikeaselkoinen [15, s. 303]. Edellä mainittujen lisäksi historiallisesti merkittäviä ja siten maininnan arvoisia ovat muun muassa Virtual Synchrony -replikointi sekä Viewstamped-replikointi. On myös olemassa lukuisia pessimistisiteen replikointiin pohjautuvia järjestelmiä, kuten esimerkiksi Google Chubby, Apache ZooKeeper ja ISIS-toolkit.

3.5.3 Optimistinen replikointi

Optimistisessa replikoinnissa tietovarantokopioiden välillä sallitaan ajoittainen epäeheys ja tästä seuraava mahdollinen konfliktien syntyminen rinnakkaisten tiedonkäsittelyoperaatioiden välillä silloin, kun järjestelmä on osittuneessa tilassa. Sanalla optimistinen kuvataan oletusta siitä, että järjestelmän todellisessa toiminnassa epäeheydestä huolimatta konflikteja ei kuitenkaan esiinny kovin usein, jos koskaan. Tietovarantokopiota käyttävän ohjelmiston oletetaan siis voivan käyttää tietovarantoa suurimman osan ajasta samoin, kuin jos tietovarantoja olisi vain yksi kappale.

Optimistisella lähestymistavalla on useita etuja pessimistiseen replikointiin verrattuna. Yhtenä tärkeimmistä järjestelmän saatavuus kyetään säilyttämään myös solmujen välisten verkkoyhteyksien katketessa. Verkon rakennetta ja järjestelmään sisältyvien solmujen joukkoa voidaan lisäksi muokata joustavammin, koska mitään tietoa esimerkiksi solmujoukon enemmistöstä ei tarvitse muodostaa jatkuvan konsensuksen saavuttamiseksi. Epäeheyden salliminen mahdollistaa tietosisältöpäivitysten asynkronisen kommunikoinnin solmujen välillä taustaprosessina, mikä helpottaa suorituskyvyn ylläpitämistä myös suurella solmujoukolla. Epäeheys mahdollistaa myös solmujen itsenäisen toiminnan siten, että järjestelmän käyttämistä voidaan jatkaa myös tietoverkosta irtautuneena. Huonona puolena optimistinen replikointi edellyttää, että järjestelmä kykenee toimimaan mielekkäästi myös epäeheyden vallitessa ja tarvittaessa ratkaisemaan rinnakkaisista tiedonkäsittelyoperaatioista aiheutuneet konfliktit eheyden palauttamiseksi. [18, s. 43-44]

Tiedonkäsittelyoperaatioiden välisillä konflikteilla voi olla huomattavia tietovarannon käyttöä monimutkaistavia vaikutuksia myös varantoa käyttävän sovelluksen näkökulmasta. Konfliktien ratkaisulogiikka muodostuu tällöin osaksi ohjelmiston yleistä tiedonkäsittelylogiikkaa. Yksinkertaisimmillaan konfliktit voidaan koettaa ratkaista auto-

maattisesti jonkin yksinkertaisen säännön perusteella, kuten aikaleimapohjaisesti siten, että ajallisesti viimeisin operaatio ylikirjoittaa muut. Tällainen lähestymistapa ei kuitenkaan takaa ratkaisun mielekkyyttä yleisessä tapauksessa, sillä kausaliteetin puuttuessa operaatioiden väliltä, on niiden ajallinenkin järjestys tyypillisesti sattumanvarainen. Käytännössä konfliktinratkaisumekanismissa tuleekin usein huomioida tiedonkäsittely-operaatioiden semantiikka. Tietovarantoa käyttävä sovellus voi esimerkiksi määritellä sen toimialalogiikkaan soveltuvan semantiikan huomioivan ratkaisulogiikan, joka kytkeytyy osaksi replikoinnin automaattista konfliktienratkaisumekanismia. Ikävä kyllä on kuitenkin mahdollista, että konfliktinratkaisu on operaatioiden semantiikkakin huomioiden liian vaikeaa, suorituskyvyllisesti kallista tai jopa mahdotonta. Ratkaisua ei pystytä suorittamaan automaattisesti esimerkiksi silloin, kun konfliktissa olevien tietojen pohjalta on jo tehty järjestelmän ulkopuolisia toimenpiteitä. Jos esimerkiksi samalta tililtä on nostettu rahaa kahdessa pankin toimipisteessä verkkoyhteyksien ollessa poikki, on mahdollista, että tilin kokonaisrahamäärä päätyy nostojen ansiosta miinukselle. Tähän ongelmaan ei ole helppoa ohjelmallisesta ratkaisua, vaan ratkaisussa tarvitaan usein toimialan sovelluksen loppukäyttäjän väliintuloa. [40, s.245]

Samaan tietoalkioon, kuten yksittäiseen pankkitiliin, kohdistuvien operaatioiden välisten suorien konfliktien lisäksi on huomioitava erilaisten järjestelmänlaajuisten invarianttien mahdollinen rikkoontuminen tietoverkon osittumisen aikana. Yksi yleinen invariantti on esimerkiksi tiettyjen tietoalkioiden yksikäsitteisyyden vaatimus koko järjestelmän tasolla. Esimerkiksi joukkotietoja käsittelevän johtamisjärjestelmän toiminta voi vaatia, että jokaisella yksittäistä joukkoa edustavalla tietoalkiolla on koko järjestelmän tasolla yksikäsitteinen tunnus. Optimistinen replikointi sallii kuitenkin sen, että kahteen tietovarantokopioon voidaan syöttää rinnakkaisesti kaksi erillistä, mutta saman tunnuksen omaavaa tietoalkiota rikkoen näin ollen invariantin. Ongelma havaitaan usein vasta kun tietoverkko palautuu osittuneesta tilasta takaisin yhtenäiseksi. [41, s. 25-26].

Tyypillinen optimistisen replikoinnin toteutus pyrkii saavuttamaan eheyden ajan myötä eheä -mallin mukaisesti siirtäen operaatioita solmujen välillä taustalla, havaiten konfliktit esimerkiksi vektorikellon avulla ja ratkaisten ne jollakin menetelmällä. Muitakin vaihtoehtoja on kuitenkin olemassa.

Yksi verrattain uusi optimistisen replikoinnin toteutustapa perustuu ajatukseen siitä, että replikoitu tietorakenne suunnitellaan alun perin sellaiseksi, että kaikki siihen tehtävät kirjoitusoperaatiot ovat kommutatiivisia eli suoritusjärjestykseltään merkityksettömiä. Kun kommutatiiviset operaatiot suoritetaan kaikkiin kopioihin, syntyy niiden välille eheys automaattisesti ilman konfliktien vaaraa. Tällaisista tietorakenteista rakenteista käytetään termiä CRDT, Commutative Replicated Data Type. CRDT-lähestymistavan ongelmana on se, että sitä ei voida käyttää kaikissa tapauksissa, sillä kaikkia tietorakenteita ei voida suunnitella ainoastaan kommutatiivisilla kirjoitusoperaatioilla muokattavaksi. [42, s. 3]

Toinen samankaltaista ajatuskulkua seuraava optimistisen replikoinnin periaate on nimeltään CALM (Consistency And Logical Monotonicity). Tämän periaatteen ajatuksena on, että ohjelmisto suunnitellaan niin sanotusti monotoniseksi. Monotonisen ohjelmiston tiedonkäsittelyoperaatiot tuottavat tuloksensa inkrementaalisesti siten, että tuloksilla ei ole koskaan vaikutusta aiempien operaatioiden tuloksiin. Tällöin saavutetaan riippumattomuus operaatioiden suoritusjärjestyksestä ja eheyden saavuttaminen tietorakennekopioiden välillä muodostuu suoraviivaiseksi. Jos ohjelmiston tarvitsee suorittaa ei-monotonisia tiedonkäsittelyoperaatioita, tulee ne suorittaa erikseen koordinoituina kaikkien kopioiden välillä jotakin konsensusprotokollaa, kuten Paxos tai 2PC käyttäen. CALM-periaatteen noudattamisen helpottamiseksi on kehitetty erillinen Bloom-ohjelmointikieli, joka ohjaa ohjelmistokehittäjää tekemään ohjelmistosta CALM-periaatteen mukaisen. [43, s. 249-250]

3.6 Optimistinen rinnakkaisuuden hallinta palvelukeskeisessä arkkitehtuurissa

Yksi optimistisen rinnakkaisuuden hallinnan muoto on usein käytössä palvelukeskeistä arkkitehtuuria noudattavissa sovelluksissa. Kun kaksi asiakassovellusta lukee palvelukutsun välityksellä saman tiedon tietokannasta ja suorittaa sitten tietoon kohdistuvan kirjoitusoperaation, on mielekästä, että jälkimmäisen kirjoitusoperaation kohdalla havaitaan aiemman operaation suoritus ja kyetään välttämään aiemman operaation tekemien muutosten ylikirjoittaminen. Jos tällaista havaitsemismekanismia ei ole käytössä, valikoituu voimaan jäävä kirjoitusoperaatio sattumanvaraisesti palvelukutsujen saapumisjärjestyksen mukaan.

Ongelma ratkaistaan tyypillisesti niin sanotun optimistisen lukituksen avulla (engl. optimistic locking). Ratkaisu toimii siten, että tietokantaan tallennettuun tietoon on liitetty erillinen versionumeroattribuutti, jota kasvatetaan jokaisen kirjoitusoperaation yhteydessä juuri ennen päivitetyn tiedon kantaan viemistä. Lisäksi kun kirjoitusoperaatiota suoritetaan, tarkistetaan ensimmäiseksi onko kannassa oleva versionumero sama kuin nyt suoritettavan operaation mukana saapuva versionumero. Jos versionumero on sama, tiedetään että samaan tietoon ei ole kohdistunut muita kirjoitusoperaatioita viimeisen lukuoperaation jälkeen. Jos versionumero ei ole sama, on jokin muu kirjoitusoperaatio ehtinyt päivittää tietoa ja ylikirjoittamisen sijasta voidaan asiakassovellukselle ilmoittaa asiasta esimerkiksi heitetyn poikkeuksen avulla. Sama logiikka voidaan toteuttaa myös käyttämällä versionumeron sijasta yksinkertaista tallennushetken aikaleimaa. Optimistinen oletus on, että ristiriitoja samanaikaisten kirjoitusoperaatioiden välillä ilmenee harvoin ja siten suurimman osan ajasta kirjoitusoperaatiot suoritetaan ongelmitta. [44, s. 90; 45]

Tämän kaltainen versionumeron ylläpitäminen voidaan nähdä vektorikellon yksinkertaistettuna muotona, joka sisältää vain yksittäisen solmun/prosessin laskuriarvon.

4 REPLIKOINTIKOMPONENTIN SUUNNITTELU

4.1 Replikointiratkaisun piirteet

Luvussa kaksi esitettyjen johtamisjärjestelmäympäristön asettamien vaatimusten seurauksena tässä työssä suunniteltava replikointiratkaisu toteutetaan seuraavien suuntalinjojen mukaisesti.

Järjestelmän korkeatasoinen vikasietoisuusvaatimus on ensisijainen suunnitteluperuste. Järjestelmän toiminta ei saa keskeytyä tietoliikenneyhteyksien katketessa tai tietovarantokopioita ylläpitävien solmujen tuhoutuessa. Tästä syystä moni-isäntä-ratkaisu on ainoa mielekäs valinta. Isäntä–renki-ratkaisussa kirjoitusoperaatioiden tekeminen vaatisi aina toimivat tietoliikenneyhteydet isäntäsolmuun, mitä ei siis voida taata. Moni-isäntäratkaisu kykenee säilyttämään toimintakykynsä myös osittuneessa/saarekkeisessa verkossa, kunhan mikä tahansa solmuista on saavutettavissa.

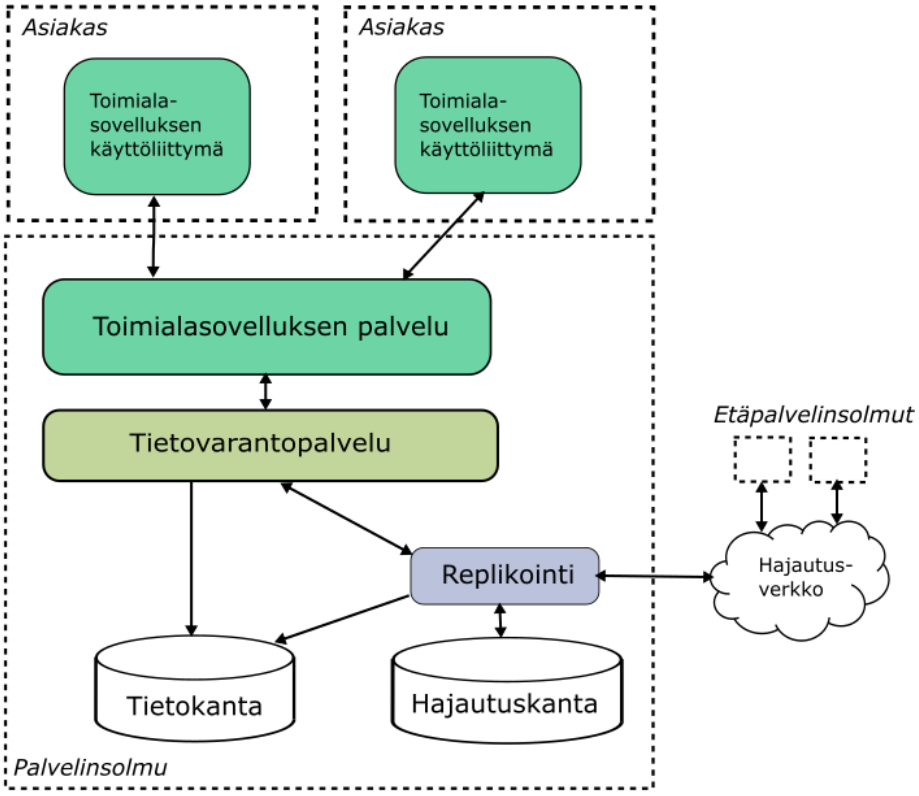
CAP-teoreeman näkökulmasta pyritään säilyttämään A- ja P-ominaisuudet, siis saatavuus ja osittumisen sietokyky. Eheyttä ei voida tällöin kaikissa tapauksissa saavuttaa, joten järjestelmän pyrkimys eheän tietosisällön muodostamiseen tulee toteuttaa ajan myötä eheä -mallin mukaisesti. Käytännössä suorituskyvyn takaamiseksi tiedonkäsittelyoperaatioiden välittäminen solmujen välillä kannattaa tehdä asynkronisesti taustaprosessina myös silloin, kun verkko ei ole osittuneessa tilassa.

Ajan myötä eheä -mallin valinta tarkoittaa, että replikointiratkaisusta tulee optimistinen. Koska toteutettava ratkaisu tulee osaksi yleiskäyttöistä alustaratkaisua, jota käyttävien toimialasovellusten tulee saada suunnitella tallennettavat tietorakenteensa vapaasti, konflikteja ei voida välttää esimerkiksi rajoittumalla CRDT-tietorakenteiden käyttöön. Näin ollen tarvitaan konfliktien havaitsemis- ja ratkaisumekanismit. Havaitsemisessa luonteva ratkaisu on kuhunkin tietosisältöpäivitykseen liitettävä oliokohtainen vektorikello. Konfliktien ratkaisemiseksi alusta tarjoaa yksinkertaisen aikaleimapohjaisen ratkaisulogiikan, jossa kellonajallisesti viimeinen operaatio yli kirjoittaa muut operaatiot. Tämä ratkaisu on hyvin rajallinen jo senkin takia, että solmujen tulee olla kellonajoiltaan synkronoituja ja ratkaisun semanttisesta mielekkyydestä ei voida olla varmoja. Tästä syystä sovelluksille tarjotaan rajapinta, jonka avulla ne voivat itse toteuttaa semanttisesti mielekkään konfliktinratkaisulogiikan. Niin halutessaan sovellukset voivat välittää konfliktin tämän rajapinnan kautta myös loppukäyttäjän ratkaistavaksi.

4.2 Arkkitehtuuri

Edellisessä aliluvussa kuvatut piirteet omaava replikointiratkaisu koostetaan useasta luku- ja kirjoitusoperaatioita tukevasta isäntäpalvelinsolmusta, joista kukin ylläpitää omaa tietosisältökopiotaan. Solmujen väliset yhteydet voivat olla ajoittain poikki, jolloin yhteyksien palatessa solmut vaihtavat niissä yhteyskatkoksen aikana tehtyjä tiedonkäsittelyoperaatioita keskenään. Kommunikaatio solmujen välillä tapahtuu kaikissa tapauksissa taustasäikeissä asynkronisesti erillään varsinaisista toimialasovellusten tekemistä kirjoitusoperaatioista.

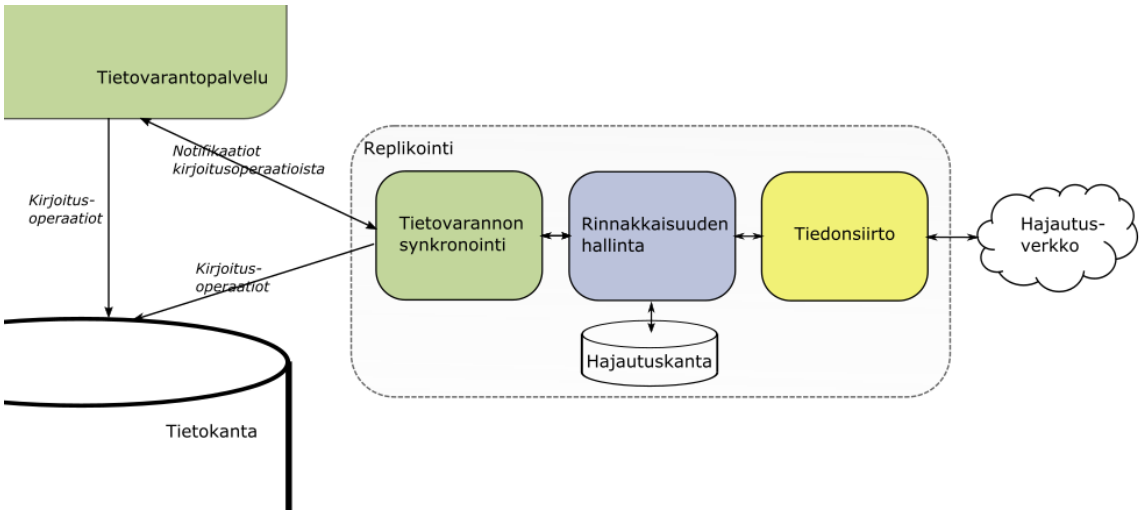
Kuva 5 esittää replikointikomponentin loogisen sijoittumisen tietovarantopalvelun taustalle ja tiedonkulun eri komponenttien välillä yksittäisessä tietosisältökopiota ylläpitävässä palvelinsolmussa. Replikointikomponentti kuuntelee toimialasovelluksen tietovarantopalvelun kautta tekemiä tietosisältöpäivityksiä ja välittää niitä edelleen hajautusverkkoon. Vastaavasti se vastaanottaa muissa hajautusverkossa toimivissa solmuissa tehtyjä päivityksiä ja välittää ne paikallisen solmun tietokantaan. Päivityksistä ilmoitetaan myös tietovarantopalvelulle niin, että ne voidaan välittää edelleen toimialasovellukselle ja sitä kautta esimerkiksi käyttöliittymäsovellukselle. Keskeinen suunnitteluperiaate on, että replikointikomponentin toiminta ei näy tietovarantopalvelun toiminnassa millään lailla, vaan se toimii näkymättömästi normaalin tiedonkäsittelytoiminnallisuuden taustalla.



Kuva 5 Replikointikomponentin looginen sijoittuminen arkkitehtuuriin ja tiedonkulku.

Yksi replikointikomponentin piirre on, että kaikki replikoitavaksi vietävä ja replikoinnin kautta vastaanotettu tietosisältö tallennetaan erilliseen hajautustietokantaan. Ratkaisulla on tiettyjä toiminnallisia ja suorituskyvyllisiä etuja. Erillinen hajautuskanta mahdollistaa ensinnäkin sen, että solmu voi toimia täysiverisenä replikointisolmuna myös sellaisten toimialasovellusten tietosisällölle, joita ei ajeta kyseisessä solmussa. Palvelinsolmun pääasiallinen tietokanta ei tällöin sisällä mitään tällaisen sovelluksen tietosisältöä. Hajautuskanta puolestaan voi sisältää, jolloin jokin toinen toimialasovellusta ajava solmu voi myös vastaanottaa tietosisältöpäivityksiä tältä solmulta. Toinen erillisen hajautuskannan etu on se, että replikointi ei kuormita palvelimen varsinaista tietokantaa. Kun yksi solmu pyytää toiselta solmulta uusia päivityksiä, kohdistuu lukuoperaatio erilliseen hajautuskantaan, joten myös pääasiallinen kuormitus kohdistuu siihen.

Replikointikomponentti koostuu kolmesta korkean tason alikomponentista, joilla kullakin on oma vastuualueensa (kuva 6). Komponentit ovat tietovarannon synkronointikomponentti, rinnakkaisuuden hallintakomponentti ja tiedonsiirtokomponentti. Näiden vastuualueet on kuvattu tarkemmin seuraavissa aliluvuissa. Näistä komponenteista hajautuksen toiminnan kannalta keskeisin on rinnakkaisuuden hallintakomponentti, jonka keskeiset algoritmit on kuvattu aliluvussa 4.3.



Kuva 6 Replikointikomponentin rakenne.

4.2.1 Tietovarannon synkronointi

Tietovarannon synkronointikomponentti kytkee tietovarantopalvelun replikointiin välittäen tiedonkäsittelyoperaatioita tietovarantopalvelusta replikoitavaksi ja replikoitua tietoa tietovarantoon. Replikointikomponentin muut osat ovat yleiskäyttöisiä, mutta synkronointikomponentti on suunniteltu kytkeytymään nimenomaan osaksi tietokannan ja

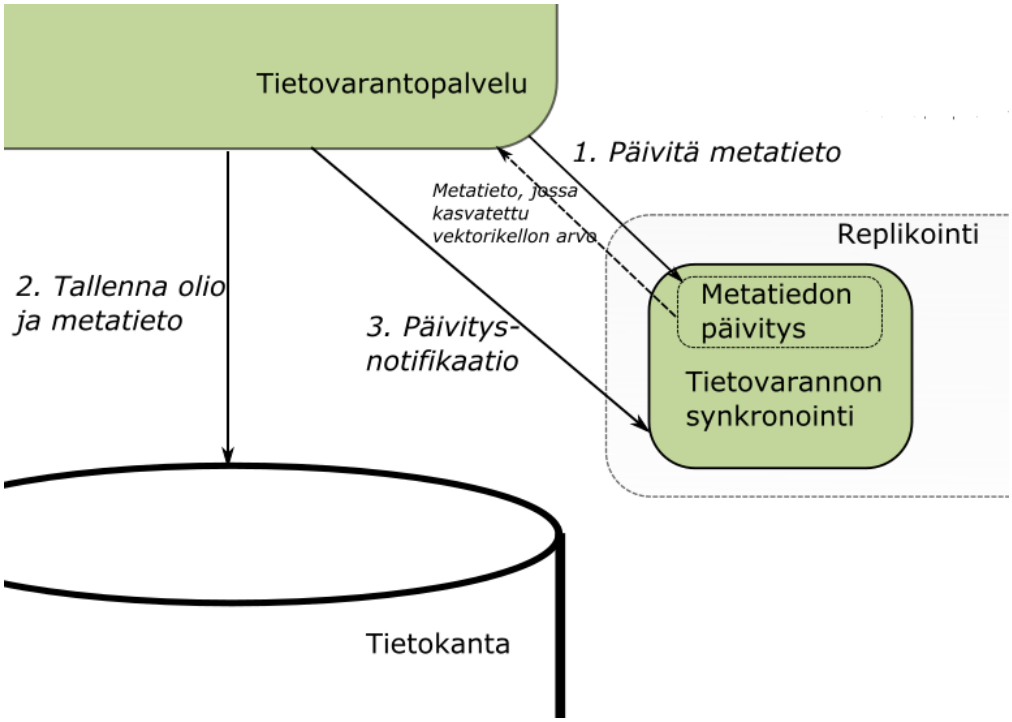
tietovarantopalvelun välistä tiedonvälitystä. Synkronointikomponentti ymmärtää, miten tietokantaan kohdistuvia kirjoitusoperaatioita kuunnellaan ja miten tietokantaan kirjoitetaan eristäen muun replikointikomponentin toteutuksen näiltä yksityiskohdilta.

Tietovarannon synkronointikomponentin vastuualueisiin kuuluvat tietovarantopalvelun julkaisemien, tietosisältöolioiden kirjoitusoperaatioista kertovien notifikaatioiden kuunteleminen sekä niiden välittäminen replikoitavaksi erityisinä hajautuspaketteina. Vastaavasti replikoinnista vastaanotetut hajautuspaketit puretaan ja niiden mukana kulkevat tietosisältöoliot tallennetaan tietokantaan sekä välitetään tietovarantopalvelulle notifikaatioina. Toimialasovellus voi konfiguroida, mitä tietokantaan tallennettavia tietosisältöolioita se haluaa replikoitavan määrittelemällä replikoitavien olioiden luokat. Yksittäinen hajautuspaketti koostuu seuraavista elementeistä: tietosisältöolion tunniste, olion luokka, päivityksen aikaleima (sen solmun ajassa, jossa päivitys on tehty), päivityksen tyyppi, vektorikello sekä tavutaulukoksi sarjallistettu tietosisältöolio. Päivitystyyppinä voi olla joko lisäys-, poisto- tai muokkausoperaatiota merkitsevä arvo. Poistotyyppisillä päivityksillä tavutaulukko on tyhjä. Kuva 7 esittää hajautuspaketin rakenteen yhtenä kokonaisuutena.

Kuva 7 Hajautuksen käsittelemän hajautuspaketin rakenne.

Hajautuspaketin eri kenttien arvot saadaan koostettua seuraavasti. Notifikaation mukana tuleva tietosisältöolio toimii itsessään tunnisteen, luokan ja tavutaulukon lähteenä. Aikaleiman arvon kertoo käyttöjärjestelmä. Notifikaatio kertoo myös päivitystyypin. Jäljelle jää vektorikellon arvon muodostaminen. Vektorikellon arvo luodaan (tai päivitetään) erikseen jo siinä vaiheessa, kun tietosisältöoliota ollaan tallentamassa varsinaiseen tietokantaan. Vektorikello myös tallennetaan varsinaiseen tietokantaan osaksi tietosisältöolion metatietoja. Tämä logiikka toteutetaan siten, että tietovarannon synkronointikomponentti rekisteröi tietovarantopalveluun erillisen metatietojen käsittelykomponentin, jolle tarjoutuu mahdollisuus muokata tallennettavan olion metatietoja ennen tietokantaan viemistä (kuva 8). Mikäli vektorikelloa ei vielä ole metatiedoissa, on kyseessä lisäysoperaatio, jolloin vektorikello luodaan paikallisen solmun laskurin saadessa arvon yksi ja muut arvon nolla. Muussa tapauksessa kyseessä on joko päivitys tai poistooperaatio ja paikallisen solmun laskuria kasvatetaan yhdellä. Vektorikelloon on näin

merkitty tieto kyseisessä palvelinsolmussa tehdystä päivityksestä jo ennen kuin se hetken päästä saapuu notifikaation mukana uudelleen tietovarannon synkronointikomponentille ja viedään edelleen replikoitavaksi.



Kuva 8 Vektorikellon päivittäminen osana tietosisältöolion tallennusoperaatiota.

Yksi synkronointikomponentin vastuista on niin sanottu alkusynkronointi. Jos palvelinsolmussa konfiguroidaan käyttöön jonkin tietosisältöoliotyypin replikointi, tulee tietokantaan mahdollisesti aiemmin tallennetut oliot hakea tietokannasta ja välittää replikoitavaksi sovelluksen käynnistyksen yhteydessä. Vastaavasti tietokannan suuntaan tuodaan muilta solmuilta saatavissa olevat tietosisältöpäivitykset, jolloin varsinainen tietokanta ja hajautuskanta on saatu synkronoitua mielekkääseen alkutilanteeseen.

4.2.2 Rinnakkaisuuden hallinta

Rinnakkaisuuden hallintakomponentti toteuttaa rinnakkaisten tietosisältöpäivitysten havaitsemiseen ja konfliktien ratkaisuun tarvittavan logiikan. Tämän komponentin suunnittelu ja toteutus on ollut erityisesti tämän diplomityön kirjoittajan vastuulla ja sen toimintaa tarkastellaan komponenteista tarkimmalla tasolla. Tämä luku kuvaa komponentin yleisen roolin kokonaisarkkitehtuurissa. Aliluku 4.3 avaa komponentin toteuttamia algoritmeja tarkemmalla tasolla.

Rinnakkaisuuden hallintakomponentin vastuualueisiin kuuluu saapuvien päivitysten vastaanottaminen sekä paikallisen palvelinsolmun tietokannan suunnasta että tiedonsiirron välityksellä muilta palvelinsolmuilta. Termi päivitys kattaa tässä kuvauksessa myös lisäys- ja poistamisoperaatiot. Saapuvat päivitykset tallennetaan erilliseen hajautustieto-

kantaan. Tämä tietokanta ylläpitää hajautuksen näkemystä siitä, mikä järjestelmän tietosisällön tila on tässä palvelinsolmussa nykyhetkellä. Kun päivityksiä saapuu, verrataan niiden mukana kulkevia vektorikelloja mahdollisesti hajautuskannassa jo olevien päivitysten vektorikelloihin, ja havaitaan näin ovatko päivitykset konfliktissa keskenään. Normaalitilanteessa, jossa konflikteja ei ole havaittu, tietokanta sisältää kustakin tietosisältöoliosta sen viimeisimmän vastaanotetun päivitysversion. Uusi saapuva päivitys korvaa aiemman, mikäli konfliktia päivitysten välillä ei havaita.

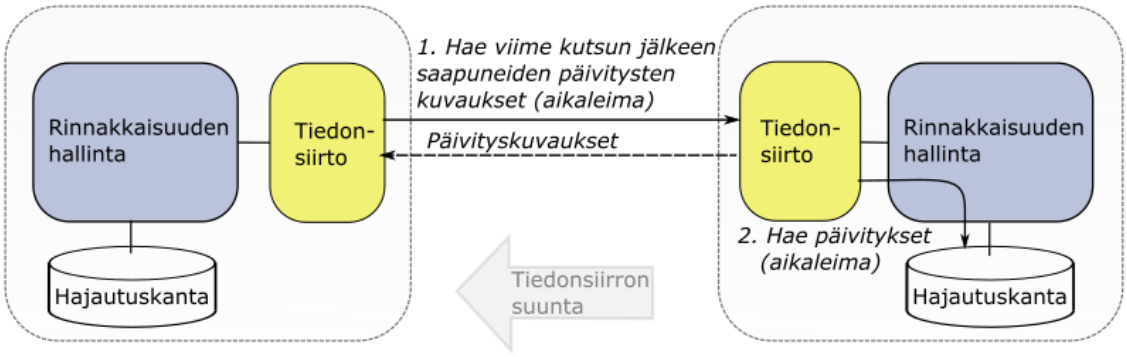
Mikäli konflikteja havaitaan, tallennetaan myös konfliktin aiheuttaneet päivitykset kantaan. Tämän jälkeen kaikki konfliktissa olevat päivitykset annetaan toimialasovelluksen konfiguroimalle konfliktin ratkaisijalle ratkaistavaksi. Ajatuksena tällaisessa useiden konfliktissa olevien päivitysten tallentamisessa on se, että mikäli kyseisessä solmussa ei pystytä ratkaisemaan konfliktia, välittyvät kaikki konfliktissa olevat päivitykset hajautuskannasta edelleen muille solmuille ja konflikti voi näin ollen tulla ratkaistuksi jossain muussa solmussa. Tilanne vastaa perinteisen versionhallinnan tilannetta, jossa on olemassa kaksi konfliktissa olevaa haaraa, joita ei ole yhdistetty. Kun konfliktiin saadaan ratkaisu ratkaisijalta, käsitellään se siten, että ratkaisusta muodostetaan uusi päivitys, jossa konfliktissa olleiden päivitysten vektorikellot on yhdistetty. Ratkaisu tallennetaan hajautuskantaan ja konfliktissa olleet päivitykset poistetaan.

Rinnakkaisuuden hallintakomponentti tarjoaa tiedonsiirtokomponentille erillisen rajapinnan, joka mahdollistaa hajautuskantaan tallennettujen päivitysten viemisen myös muille solmuille. Näihin lukeutuvat solmussa paikallisesti tehdyt päivitykset sekä muilta solmuilta aiemmin vastaanotetut päivitykset.

4.2.3 Tiedonsiirto

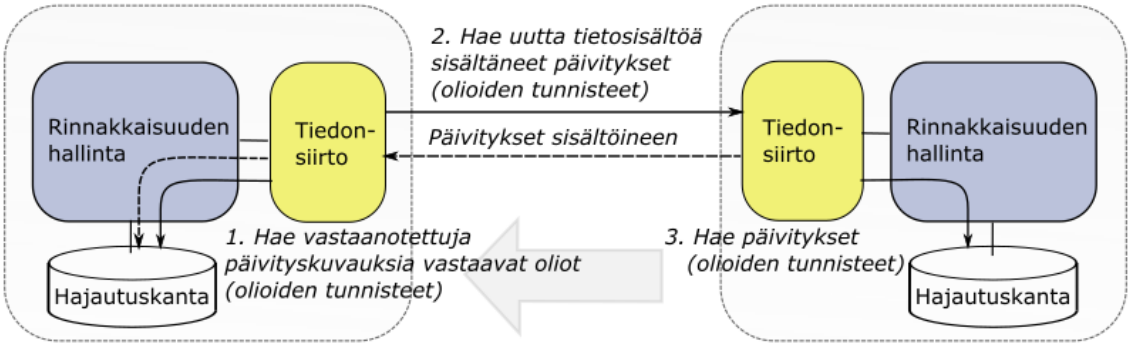
Tiedonsiirtokomponentti vastaa nimensä mukaisesti tiedon siirtämisestä ja vastaanottamisesta muilta hajautusverkon solmuilta. Mahdollisia toteutustekniikoita on useita. Hajautukseen tietokannan suunnasta saapuvia tietosisältöpäivityksiä voitaisiin esimerkiksi kuunnella ja lähettää hajautusverkkoon multicast-osoitteistuksella. Koska johtamisjärjestelmäympäristössä käytössä olevat tietoliikennetekniikat ovat hyvin vaihtelevia – esimerkiksi radioyhteyksien ylitse ei IP-pohjaista multicast-osoitteistusta ole mahdollista välttämättä käyttää – on tiedonsiirto kuitenkin helpointa toteuttaa pollaukseen perustuen. Hajautusverkon solmut siis kysyvät toisiltaan toistuvasti tietyn ajanjakson välein mahdollisesti saatavilla olevia uusia päivityksiä. Tiedonsiirron toimintalogiikan kuvaus jätetään tässä työssä tietoisesti korkealle tasolle. Esimerkiksi se, miten palvelinsolmut löytävät toisensa hajautusverkosta, jätetään kuvaamatta. Oletuksena on, että verkossa sijaitsevat solmut ovat kyvykkäitä tekemään etäkutsuja toisilleen jollakin käytössä olevalla tiedonsiirtoprotokollalla. Tiedonsiirto hyödyntää omassa toiminnassaan kuitenkin myös rinnakkaisuuden hallinnan mekanismeja kelvollisen suorituskyvyn saavuttamiseksi, joten toimintalogiikka on mielekästä kuvata yleisellä tasolla.

Tietosisältöolioiden päivitysten kyselykutsut yksittäiseltä etäsolmulta suoritetaan kaksivaiheisesti. Kuva 9 kuvaa ensimmäisen vaiheen. Etäsolmulta kysytään yksinkertaiset kuvaukset niistä päivityksistä, jotka ovat saapuneet pollauksen kohteena olevaan solmuun joko paikallisesta tietokannasta tai muilta solmuilta edellisen kyselykutsun jälkeen. Kuvaus sisältää kunkin päivityksen kohteena olleen tietosisältöolion tunnisteen, päivityksen tehneen solmun tunnisteen sekä päivitykseen liittyvän vektorikellon arvon. Jos edeltävää kutsua ei ole tehty, vaan kyselyä suorittava solmu on vasta liittynyt hajautusverkkoon, saadaan vastaukseksi kuvaukset kaikista hajautuskannassa olevista päivityksistä. Loogisesti tämä vastaa tietynlaista etäsolmun alkusynkronointia yksittäisen etäsolmun kanssa.



Kuva 9 Tiedonsiirron ensimmäinen vaihe. Päivityskuvausten haku.

Toisessa vaiheessa (kuva 10), päivitysten kuvaukset saatuaan, solmu tarkistaa ensin mitkä päivitykset todella tarvitsee ladata etäsolmulta. Ensimmäisenä tarkastetaan onko vastaanotetun päivityksen tehnyt solmu päivitystä lataava solmu itse. Jos on, ei tehtyä päivitystä tarvitse enää luonnollisesti ladata takaisin tähän solmuun. Seuraavaksi tarkastetaan vektorikellon avulla sisältääkö päivitys tämän solmun näkökulmasta uutta tietosisältöä. Tämä tehdään siten, että kysytään rinnakkaisuuden hallintakomponentilta onko vastaanotetun päivityksen vektorikellon arvon omaava päivitys tai jokin siihen kausaalisen suhteen omaava uudempi päivitys jo tallennettuna paikalliseen hajautuskantaan. Jos on, tiedetään että latauksen kohteena oleva päivitys on tämän solmun tietosisällön näkökulmasta vanhentunutta tietoa. Kun kaikki kuvaukset käydään tällä tavoin lävitse, muodostetaan lista päivityksistä, joiden varsinainen sisältö ladataan etäsolmulta.



Kuva 10 Tiedonsiirron toinen vaihe. Uutta tietosisältöä sisältävien päivitysten haku ja tallennus.

Koska etäsolmulta päivityksiä pyydettäessä palautetaan niin paikallisesti kyseisessä etäsolmussa tehdyt päivitykset, kuin myös muilta etäsolmuilta vastaanotetut päivitykset, leviävät päivitykset tehokkaasti hajautusverkossa riippumatta verkkotopologiasta. Tällaisesta tietoa asteittain levittävästä tiedonsiirtologiikasta käytetään usein termiä juoru protokolla (engl. gossip protocol) [46]. Vaikka hajautusverkko jakautuisi itsenäisiin saarekkeisiin, riittää tietosisällön eheytymiseen koko järjestelmässä se, että jossakin vaiheessa kaksi eri saarekkeissa sijaitsevaa solmua muodostavat yhteyden toisiinsa ja kykenevät vaihtamaan tiedossaan olevia päivityksiä keskenään. Kaikki toisessa saarekkeessa tehdyt päivitykset leviävät lopulta myös kaikkiin toisen saarekkeen solmuihin. Verkkotopologia voi olla myös ketjumainen tai muodostaa silmukoita. Edellä kuvatulla kaksivaiheisella vektorikelloa hyödyntävällä tiedonsiirtologiikalla saavutetaan näin ollen toimintaympäristön vaatimukset täyttävä ja ainakin kohtuullisen tehokas tiedonsiirtomekanismi.

4.3 Rinnakkaisuuden hallinnan toimintalogiikka

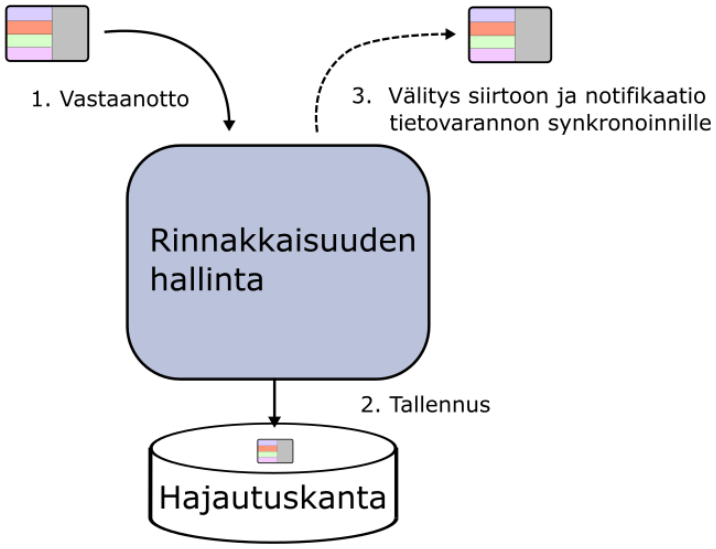
Rinnakkaisuuden hallintakomponentti ja sen ylläpitämä hajautuskanta säilyttävät järjestelmän tilaa hajautuksen näkökulmasta. Hajautuskantaan vastaanotetaan niin paikallisessa solmussa toimialasovelluksen tekemiä päivityksiä, kuin myös muilta solmuilta siirrettyjä päivityksiä. Vastaavasti hajautuskannasta viedään päivityksiä paikalliseen kantaan sekä muille solmuille tiedonsiirron välityksellä. Rinnakkaisuuden hallintakomponentin toiminta perustuu hajautuspaketteihin, jotka luodaan tietovarannon synkronointikomponentin toimesta tietovarannolta saapuneen notifikaation sisällöstä. Hajautuspakettiin sisältyvä päivitystyyppi ratkaisee sen mikä seuraavissa aliluvuissa kuvatuista paketin käsittelytavoista suoritetaan. Saapuessaan rinnakkaisuuden hallintakomponentin käsiteltäväksi hajautuspaketin mukana kulkevaa vektorikellon arvoa on myös muokattu siten, että päivityksen tehnyttä solmua vastaavan laskuritaulukon arvoa on kasvatettu (kuva 8).

Hajautuspakettien käsittelylogiikan kannalta on yhdentekevää saapuvatko paketit tiedonsiirron vai paikallisen tietovarantopalvelun suunnasta. Ainoa ero tietovarannon synkronointikomponentin ja tiedonsiirtokomponentin välittämien pakettien käsittelyssä

on se, että tietovarannon suunnasta tulevasta päivityksestä ei tarvitse enää hajautuskantaan tallentamisen jälkeen lähettää notifikaatiota tietovarantopalvelulle.

4.3.1 Lisäysoperaatio

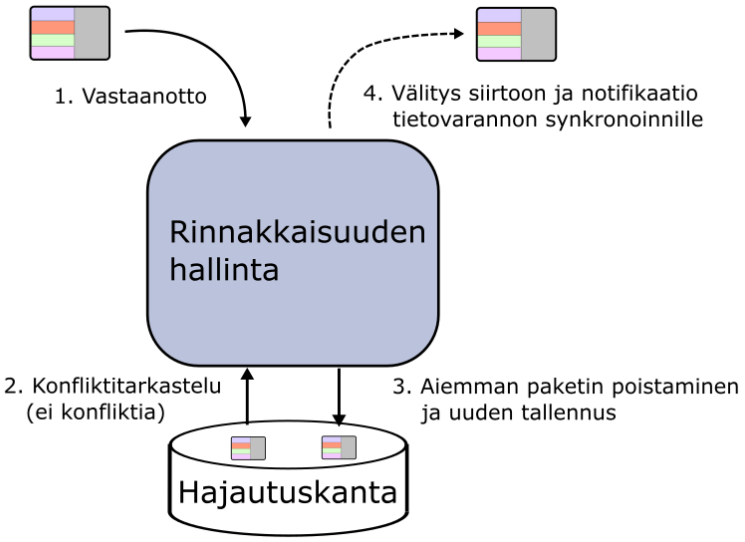
Tietosisältöolion luominen tuo rinnakkaisuuden hallintakomponentille uuden hajautuspaketin, joka ei voi olla konfliktissa minkään aiemmin hajautuskantaan tallennetun päivityksen kanssa. Tällaisen päivityksen käsittely on yksinkertaista. Paketti yksinkertaisesti tallennetaan hajautuskantaan, josta tiedonsiirtokomponentti hakee sen siirrettäväksi muihin solmuihin (kuva 11). Lisäksi muilta solmuilta vastaanotetuista paketeista annetaan notifikaatio tietovarannon synkronointikomponentille. Hajautuskantaan tallennetun paketin vektorikello sisältää vain paikallisen solmun laskurin arvon, joka on yksi.



Kuva 11 Lisäysoperaation sisältävän hajautuspaketin käsittelylogiikka.

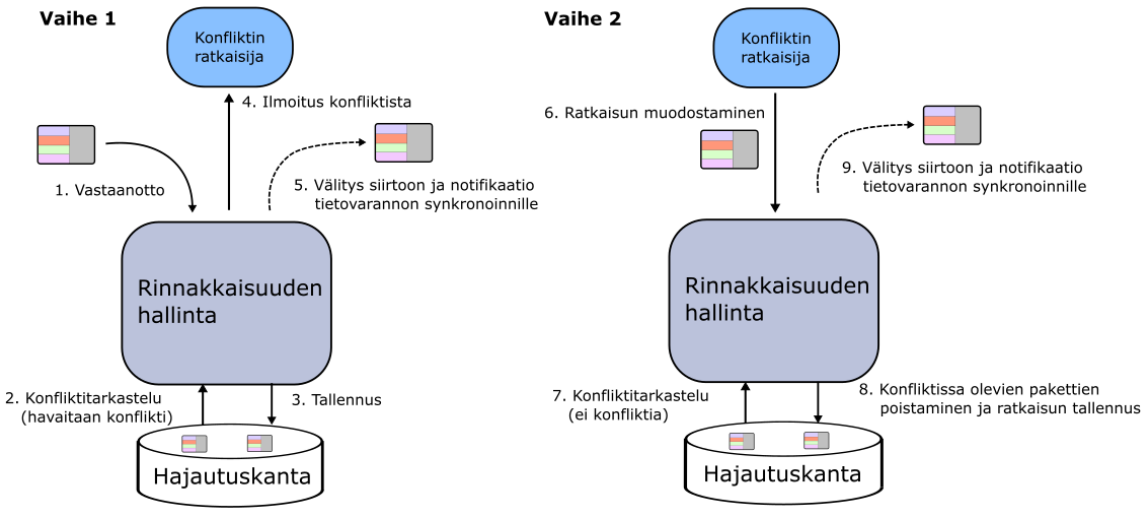
4.3.2 Muokkausoperaatio

Tietosisältöolion muokkausoperaatiosta kertovan hajautuspaketin käsittely haarautuu loogisesti kahteen vaihtoehtoiseen suuntaan sen mukaan havaitaanko konflikti aiemman hajautuskantaan tallennetun hajautuspaketin kanssa vai ei. Jos konfliktia ei havaita vektorikellojen perusteella, on vastaanotettu hajautuspaketti kausaalisessa suhteessa aiempaan, siis ”uudempi versio” samasta tietosisältöoliosta. Tällöin vastaanotettu hajautuspaketti korvaa aiemmin kantaan tallennetun paketin (kuva 12). Paketin korvaaminen vastaa aliluvussa 3.4 kuvattua vektorikellon käsittelysääntöä numero neljä.



Kuva 12 Muokkausoperaation sisältävän hajautuspaketin käsittely, kun ei havaita konfliktia.

Kun vastaanotettavan hajautuspaketin havaitaan olevan konfliktissa yhden tai useamman aiemmin hajautuskantaan tallennetun hajautuspaketin kanssa, toimitaan kaksivaiheisesti. Ensimmäisessä vaiheessa vastaanotettu paketti tallennetaan hajautuskantaan aiemman hajautuspaketin rinnalle ja ratkaisijakomponentille annetaan tieto havaitusta konfliktista. Saman tietosisältöolion tilaa kuvaavia paketteja säilytetään siis vielä tässä vaiheessa kaksi tai useampia kappaleita. Toisessa vaiheessa konfliktinratkaisija muodostaa ratkaisun ja antaa sen rinnakkaisuuden hallintakomponentille uutena hajautuspakettina. Ratkaisun sisältävä paketti käsitellään aivan kuten mikä tahansa vastaanotettu päivitys. Oleellista on, että ratkaisun sisältävän paketin mukaan liitettävä vektorikello on sellainen, että se muodostaa kausaalisen korvaajan kaikille konfliktissa olleille hajautuspaketeille. Tällöin alkuperäiset konfliktissa olleet hajautuspaketit voidaan poistaa hajautuskannasta ja tilalle tallennetaan ratkaisun sisältävä hajautuspaketti (kuva 13).



Kuva 13 Muokkausoperaation sisältävän hajautuspaketin käsittely, kun havaitaan konflikti.

Konfliktin ratkaisija muodostaa siis nyt vektorikellon käsittelysäännön numero neljä mukaisen vektorikellon arvon sekä uuden sisällön tietosisältöoliolle tavutaulukoksi sarjallistettuna. Sisältö voi olla periaatteessa mitä tahansa riippuen konfliktin ratkaisijan toteutuksesta. Se voi olla jonkin konfliktissa olleen hajautuspaketin tietosisältöolion sisältö sellaisenaan tai esimerkiksi jokin yhdistelmä pakettien sisällöistä. Sisältö voi olla myös tyhjä tavutaulukko, jos konfliktin ratkaisija toteaa, että oikea ratkaisu on tietosisältöolion poistaminen. Tällöin myös hajautuspaketin päivitystyypin arvoksi asetetaan poisto-operaatiota merkitsevä arvo.

On mahdollista, että solmussa, jossa konflikti havaitaan, ei saada ratkaisua aikaan. Konfliktin ratkaisija ei tällöin siis anna rinnakkaisuuden hallinnalle ratkaisupakettia. Syynä voi olla esimerkiksi se, ettei paketteihin sisältyviä tietosisältöolioita käsittelevää toimialasovellusta, eikä siten sen mukana konfiguroitua konfliktinratkaisijaakaan, ole asennettu solmuun, jossa konflikti havaitaan. Tällöin hajautuskantaan tallennetut, konfliktissa olevat, hajautuspaketit jäävät hajautuskantaan ja tiedonsiirto välittää ne kaikki sellaisenaan muille solmuille. Konflikti ratkeaa, kun jokin ratkaisuun kykenevä solmu lopulta vastaanottaa hajautuspaketit ja luo ratkaisupäivityksen sisältävän paketin. Tämä toimintatapa mahdollistaa sen, että hajautus toimii, vaikka tietty toimialasovellus olisi asennettu vain harvoihin solmuihin. Ylläpitäjälle tarjotaan toisaalta myös mahdollisuus konfiguroida tiedonsiirtokomponenttia niin, että vain haluttujen toimialasovellusten tietosisältöolioita ladataan kyseiseen solmuun.

4.3.3 Poisto-operaatio

Vaikka poisto-operaatio on toimialasovelluksen käyttäjän näkökulmasta hyvin erilainen toiminto muokkausoperaatioon verrattuna, hajautus käsittelee näitä operaatiotyyppejä täsmälleen samalla tavalla. Poisto-operaatio nähdään muokkausoperaationa, joka merkitsee tietyn tietosisältöolion elinkaaren päättymistä. Kun tieto vastaanotetusta poistotyyppisestä hajautuspaketista saapuu hajautuskomponentilta notifikaationa tietovarantopalvelulle, poistetaan tietosisältöolio varsinaisesta tietokannasta. Hajautuskantaan poisto-operaatiota vastaava hajautuspaketti kuitenkin jää. Tämä on tarpeen siitä syystä, että yhteyskatkojen aikana tapahtuneet poisto-operaatiot on kuitenkin edelleen kyettävä kommunikoimaan muille solmuille. Poisto-operaatiot voivat myös olla konfliktissa muokkausoperaatioiden kanssa ja ratkaisu tällaisiin konflikteihin voi olla olion ”herättäminen kuolleista”. Jos esimerkiksi jokin tietosisältöolio on poistettu tietyssä solmussa ennen kuin on saatu tietoa toisessa solmussa tehdystä muokkausoperaatiosta, joka olisi tehnyt olion säilyttämisestä tärkeää, voi olla mielekästä, että konfliktin ratkaisija palauttaa olion järjestelmään.

4.4 Toiminnallisuutta tukevia ratkaisuja

4.4.1 Poisto-operaatioiden poistaminen

Hajautus käsittelee poisto-operaatioista kertovia hajautuspaketteja samoin kuin muokkausoperaatioista kertovia paketteja. Kun tietty tietosisältöolio poistetaan jää poisto-operaatiosta kertova hajautuspaketti hajautuskantaan pysyvästi. Ongelmallista on, että näitä hajautuspaketteja kertyy hajautuskantaan ajan myötä mahdollisesti suuria määriä, mikä voi aiheuttaa suorituskyvyllisiä tai levytilan riittävyyteen liittyviä ongelmia. On mielekästä, että hajautuksen yhteydessä toimii erillinen taustaprosessi, joka käy poistamassa riittävän, tietosisältöoliotyyppikohtaisesti konfiguroitavissa olevan iän saavuttaneita poistotyyppisiä hajautuspaketteja hajautuskannasta. Tämä pitää hajautuskannan koon rajallisena, mutta aiheuttaa samalla toisen, toivottavasti pienemmän, ongelman. Jotkin hajautusverkon solmut voivat nimittäin jäädä vaille tietoa siitä, että tietty tietosisältöolio on poistettu ja sallia virheellisesti sen käytön jatkamisen toimialasovelluksessa. Tilanteen todennäköisyys kasvaa sitä mukaa, mitä kauemmin solmut tyypillisesti viettävät aikaa irrallaan muusta hajautusverkosta. Tärkeää onkin, että poisto-operaatiosta kertovien hajautuspakettien maksimi-ikä konfiguroidaan tarkkaa harkintaa käyttäen.

4.4.2 Konfliktin ratkaisujen väliset konfliktit

Edellä kuvattu rinnakkaisuuden hallinnan logiikka pitää sisällään jotakin ongelmallisia piirteitä, jotka pitää ratkaista erikseen. Yksi tällainen ongelma on eri solmuissa tehdyt rinnakkaiset konfliktinratkaisut, jotka ovat itsessään konfliktissa keskenään. Tilanne syntyy esimerkiksi silloin, kun kaksi tai useampia solmuja päivittävät samaa tietosisältöoliota ja solmut lataavat hajautuspaketit toisiltaan samanaikaisesti ristiin. Sopivalla ajoituksella solmut voivat havaita ja ratkaista saman konfliktin yhtäaikaisesti. Ratkaisu voi olla jopa erilainen, jos se teetetään esimerkiksi loppukäyttäjällä. Koska konfliktinratkaisun yhteydessä ratkaisun tekevä solmu kasvattaa lopuksi omaa vektorikellon laskuriarvoaan, tehden niistä kuin minkä tahansa solmussa tehdyn päivityksen, ovat konfliktin eri ratkaisut itsessään nyt konfliktissa. Huonolla tuurilla solmut vaihtavat ratkaisujaan uudelleen ristiin ja tätä uutta konfliktia ruvetaan ratkaisemaan jälleen useassa solmuissa niin, että tuloksena on uusi ratkaisujen välinen konflikti. Lopputulos voi olla, että vaikka alkuperäiseen konfliktiin olisi ollut saatavilla selkeä ratkaisu, konfliktinratkaisumekanismi ei koskaan pääse yksikäsitteiseen ratkaisuun päivitetyn tietosisältöolion tilasta. Ongelmalta vältytään siten, että muodostettaessa konfliktin ratkaisevaa hajautuspakettia merkitään siihen erillinen tieto siitä, että kyseessä on konfliktin ratkaisun sisältävä paketti. Jos jossain vaiheessa konfliktin ratkaisija saa ratkaistavakseen tällaisen ratkaisupakettien välisen konfliktin, käytetään jotakin järjestelmänlaajuisesti yhtenäistä logiikkaa siitä minkä solmun tuottama ratkaisu jää voimaan. Muut ratkaisut yksinkertaisesti korvataan kyseisen solmun tuottamalla ratkaisulla. Yksinkertaisimmillaan logiikka voi perustua siihen, missä järjestyksessä solmujen laskurit ovat vektorikellon laskuritau-

lukossa. Se solmu, jonka laskurin arvo sijaitsee pienemmässä taulukon indeksissä, on solmu, jonka konfliktin ratkaisu jää voimaan kahden konfliktin ratkaisun välisessä konfliktissa. Tämä yksinkertainen logiikka takaa, että lopulta kaikki solmut muodostavat yhteneväisen käsityksen siitä, mikä ratkaisu jää voimaan.

4.4.3 Sisällöltään yhteneväiset päivitykset

On mahdollista, että kaksi solmua tekevät samanlaisen päivityksen tietosisältöolioon samanaikaisesti. Hajautuksen näkökulmasta tilanteessa syntyy konflikti, joka tarvitsee ratkaista, vaikka tosiasiassa mitään sisällöllistä konfliktia ei ole syntynyt. Jotta vältyttäisiin turhan ratkaisun muodostamiselta ja siten turhalta tietoliikenteeltä, voidaan konfliktin ratkaisua tällaisessa tilanteessa yksinkertaistaa. Aluksi konfliktinratkaisija tarkastaa aina ensimmäisenä ovatko konfliktissa olevien hajautuspakettien sisällöt samanlaiset. Jotta sisällöt muodostavia tavutaulukkoja ei tarvitsisi käydä lävitse kokonaan, voidaan hajautuspakettien luomisvaiheessa laskea sisällöille tiivistekoodit (engl. hash code), jotka liitetään hajautuspaketin osaksi. Näiden koodien vertaaminen kertoo nopeasti ovatko sisällöt yhteneväiset. Jos yhteneväisyys todetaan, muodostetaan ratkaisu, jossa uusiksi vektorikellon laskuritaulukon arvoiksi valitaan konfliktissa olevien päivitysten laskurien maksimiarvot ja tavutaulukon sisällöksi jommankumman päivityksen sisältö. Ratkaisua tekevän solmun laskuria ei kuitenkaan nyt kasvateta, kuten normaalisti. Tällä on se vaikutus, että näin saatu ratkaisu ei ole koskaan konfliktissa muiden samaa logiikkaa noudattavien solmujen tuottamien ratkaisujen kanssa. Solmut päätyvät täsmälleen samaan ratkaisuun toisistaan riippumatta. Turhalta tiedonsiirrolta vältytään, kun tiedonsiirtokomponentti huomaa vektorikelloja vertaamalla paikallisen hajautuskannan jo sisältävän saman vektorikellon omaavan päivityksen.

5 RATKAISUN ARVIOINTIA JA PARANNUSEHDOTUKSIA

5.1 Arkkitehtuurin ongelmakohtia ja vaihtoehtoinen ratkaisu

Replikointiratkaisua suunniteltaessa tehtiin aikaisessa vaiheessa päätös arkkitehtuurista, jossa kaksi tietokantaa, hajautuskanta ja varsinainen toimialasovellusten käytössä oleva tietovarantopalvelun tietokanta, toimivat rinnakkain ja säilyttävät käytännössä kahta kopiota samasta tiedosta yksittäisessä solmussa (kuva 5). Vastaavasti replikointikomponentin toimiminen tietovarantopalvelurajapinnan taustalla, piilossa toimialasovelluksilta, oli yksi aikaisista arkkitehtuuriratkaisuista. Näillä ratkaisuilla replikointikomponentti saatiin eroteltua loogisesti omaksi kokonaisuudekseen ja samalla sen toiminnan vaikutukset toimialasovellusten suorituskykyyn saatiin minimoitua. Ikävä kyllä näillä ratkaisuilla on myös selkeät ongelmakohtansa, joita esitellään seuraavissa luvuissa.

5.1.1 Kahden kopion ylläpitäminen

Arkkitehtuurin perusajatus on, että tietovarantopalvelun ylläpitämästä tietokannasta luodaan paikallinen kopio, siis hajautuskanta, lähettämällä asynkronisia notifikaatioita tietosisältöolioiden päivitysoperaatioista replikointikomponentille. Vastaavasti tietovarantopalvelu vastaanottaa asynkronisia päivitysoperaatioita replikointikomponentilta ja pitää itsensä siten identtisenä hajautuskantaan nähden (ainakin siihen asti kun hajautuskantaan tallentuu konfliktissa olevia eri olioversioita). Replikoinnin tiedonsiirtokomponentin toiminta suorittaa loogisesti samanlaisen asynkronisen kaksisuuntaisen päivitysten siirron muiden solmujen hajautuskantoihin. Pohdinnan arvoinen kysymys on, miksi hajautuskantaan tallennettaessa tarvitaan sivistynyt konfliktien havaitsemislogiikka, mutta varsinaiseen tietokantaan päivityksiä tuotaessa ei? Hajautuskanta ja varsinainen tietokanta ovat kuitenkin molemmat asynkronisesti ylläpidettäviä ja rinnakkaisesti päivitysoperaatioita vastaanottavia kopioita samasta tiedosta. Jos asiaa tarkastellaan lähemmin, on selvää, että ainakin osittainen konfliktien havaitsemislogiikka tarvitsee tehdä myös varsinaiseen tietokantaan saapuville päivityksille.

Esimerkkinä voidaan käyttää tilannetta, jossa sama tietosisältöolion versio on päivittynyt alkutilanteessa kahteen eri solmuun. Toinen solmuista (etäsolmu) tekee uuden päivityksen ja päivitys saapuu paikallisen solmun varsinaiseen tietokantaan tallennettavaksi replikoinnin kautta. Normaalitilanteessa päivitys viedään tietokantaan korvaamalla siellä aiemmin ollut olion versio ja päivityksestä annetaan tietovarantopalvelun kautta noti-

fikaatio toimialasovellukselle. Ongelmia syntyy, jos samalla hetkellä, kun etäsolmulta vastaanotettua päivitystä ollaan viemässä kantaan, tehdään myös tietovarantopalvelun kautta päivitys paikallisesti. Nämä päivitykset muodostavat replikoinnin näkökulmasta konfliktin. Paikallisen päivityksen ehtiessä tietokantaan ensin korvautuu se hetimiten etäsolmulta saadulla päivityksellä. Toimialasovellus vastaanottaa ensin paikallisen päivityksen notifikaation ja sitten etäsolmulta saadun päivityksen notifikaation. Loppukäyttäjän näkökulmasta hänen juuri paikallisesti tekemänsä muutokset näyttävät tulleen peruutetuksi ja korvatuksi etäsolmun tekemän päivityksen tiedoilla. Ongelman yksi mahdollinen ratkaisu olisi tehdä vektorikellopohjainen konfliktitarkastelu myös aina tallennettaessa päivityksiä varsinaiseen tietokantaan ja suorittaa tallennus vain, jos uusi päivitys ei ole konfliktissa aiemmin kantaan tallennetun tietosisältöversion kanssa. Tämä tarkoittaa kuitenkin rinnakkaisuuden hallinnan logiikan monistamista kahteen paikkaan, mikä ei ole kovinkaan siisti ratkaisu.

Toinen varsinaisen tietokannan ja hajautuskannan ylläpitämiseen liittyvä ongelma liittyy yleisen tason luotettavuuteen. Jos yksikin hajautuskannan ja varsinaisen tietokannan välillä tietoa välittävä notifikaatio jää saapumatta tai notifikaation käsittely epäonnistuu jostain syystä, jäävät tietokannat eri tilaan. Syitä voivat olla esimerkiksi ohjelmointivirheet ja erilaiset poikkeustilanteet järjestelmän toiminnassa, kuten järjestelmän sammuttavat sähkökatkokset. Tietokantojen ollessa eri tilassa paikallisesti toimiva toimialasovellus toimii eri tietosisällön varassa kuin muut solmut, jotka saavat tietonsa hajautuskannasta. Tällaista tilannetta ei voida lähtökohtaisesti sallia ainakaan ilman minkäänlaista korjaavaa mekanismia. Yksi ehdotettu ratkaisu tähän ongelmaan on erillinen synkronointitoiminto, joka vie ensin kaikki varsinaisessa tietokannassa olevat tietosisältöoliot replikointikomponentin käsiteltäväksi ja sen jälkeen kaikki hajautuskannan sisältämät päivitykset varsinaiseen tietokantaan. Jos kantojen eroavaisuudet ovat pieniä, suurin osa välitetyistä tiedoista tulee hylätyksi vektorikellotarkastelun ansiosta (myös varsinaiseen tietokantaan päivityksiä vietäessä tarvitaan siis tässä tapauksessa edellä kuvattu vektorikellopohjainen tarkastuslogiikka, joka hylkää jo kantaan tallennetut päivitykset). Lopputuloksena molemmat kannat ovat yhteneväisiä. Tässä ratkaisussa on kuitenkin edelleen se ongelma, että on käytännössä äärimmäisen vaikea tietää milloin tällainen tietokantojen synkronointi tarvitsee tehdä. Sähkökatkosten aiheuttamana ongelma ratkeaa tekemällä synkronointi aina järjestelmän käynnistyessä, mutta tällöin operaatio suoritetaan huomattavan usein myös turhaan. Tämä on ongelma etenkin, koska synkronointioperaatio voi olla hyvinkin pitkäkestoinen ja suorituskyvyllisesti vaativa, mikäli kantojen sisältämät tietomäärät ovat suuret.

5.1.2 Invarianttien ylläpitämisen vaikeus

Aliluvussa 3.5.3 keskusteltiin optimistisen replikoinnin liittyvästä haasteesta, jossa erilaisten järjestelmänlaajuisten invarianttien ylläpitäminen on mahdotonta silloin, kun verkkoyhteydet ovat poikki. Toimialasovelluksen toimintalogiikka voi esimerkiksi vaatia, että jokaisella järjestelmään tallennettavalla joukolla on yksilöllinen tunniste. Tun-

nisteiden yksilöllisyyttä ei voida tarkistaa tallennettaessa, jos tallennus voi tapahtua useissa replikointiverkon solmuissa samanaikaisesti verkkoyhteyksien ollessa poikki. Invarianttia ei siis pystytä kaikissa tapauksissa ylläpitämään. Järjestelmän mielekkään toiminnan kannalta on kuitenkin tärkeää, että järjestelmä kykenee verkkoyhteyksien palatessa jollakin tavalla palauttamaan invariantin voimaan, jotta järjestelmässä ei ole vastaisuudessa esimerkiksi saman tunnuksen omaavia joukkoja.

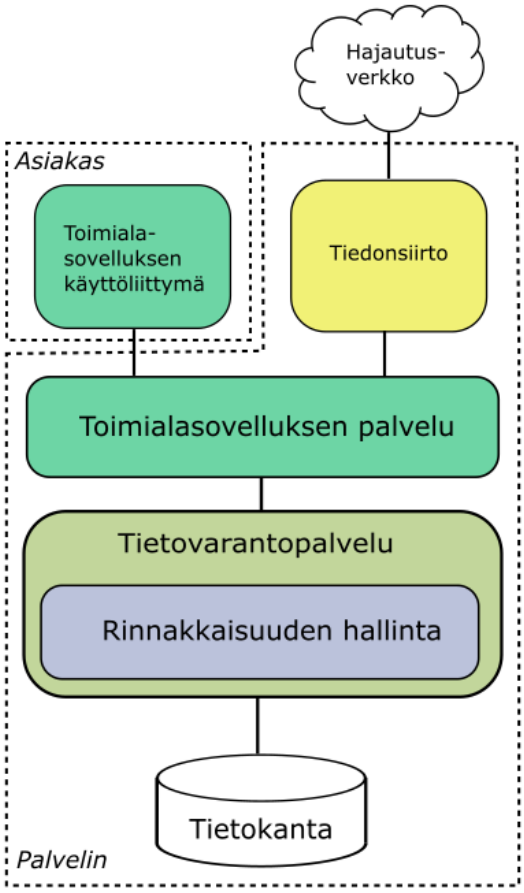
Sattumalta juuri joukkojen yksilöllisen tunnuksen kaltaisen invariantin ylläpitämiseen löytyy ratkaisu ilman erityisiä muutoksia edellä esitettyyn arkkitehtuuriin. Toimialasovellukselle voidaan tarjota mahdollisuus käyttää yksilöllistä joukon tunnistetta tietosisältöolion varsinaisena tunnisteena, siis eräänlaisena luonnollisena avaimena. Tällöin replikointikomponentti kykenee havaitsemaan saman tunnisteen omaavan joukon luomisen kahdessa eri solmussa konfliktinratkaisumekanismin avulla ja joukkojen tiedot voidaan yhdistää, tai toinen joukko voidaan poistaa osana konfliktin ratkaisua. Ikävä kyllä järjestelmän invariantit eivät välttämättä ole aina näin yksinkertaisia. On esimerkiksi mahdollista, että tietyllä maantieteellisellä alueella tulee aina sijaita tietty määrä joukkoja. Jos eri joukkojen sijainteja muutetaan kahdessa eri solmussa yhteyskatkon aikana, voi lopputulos olla, että alueella ei ole riittävää määrää joukkoja ja invariantti on rikkoontunut. Ongelmalle ei voida mitään yhteyskatkon aikana, mutta yhteyksien palatessa invariantin rikkoutuminen tulisi olla mahdollista havaita ja korjata. Replikoinnin konfliktinratkaisumekanismista ei ole apua tässä tapauksessa, koska kyseisenkaltaisen invariantin rikkoontuminen ei välttämättä aiheuta yhtään konfliktia. Replikointiratkaisun nykyisestä arkkitehtuurista puuttuu siis selvästi toiminto, joka mahdollistaisi toisista solmuista peräisin olevien päivitysten tarkastamisen ja mahdollistaisi invarianttien palauttamisen käyttäen päivitysten sisältöä lähtötietoina.

5.1.3 Vaihtoehtoinen arkkitehtuuri

Tässä aliluvussa esitetään hahmotelma arkkitehtuurista, jonka avulla voitaisiin ratkaista edellisissä luvuissa esitellyt kahden kopion sekä järjestelmänlaajuisten invarianttien ylläpitämisen ongelmat ja näkökulmasta riippuen myös yksinkertaistaa arkkitehtuuria tietyissä määrin.

Vaihtoehtoisessa arkkitehtuurissa luovutaan erillisen tietovarantopalvelun taustalla toimivan replikointikomponentin ideasta ja sovitetaan replikoinnin logiikka kiinteäksi osaksi palvelukeskeistä arkkitehtuuria (kuva 14). Käytössä on vain yksi tietokanta, jossa säilytetään tietosisältöolioita yhdessä niiden päivityshistoriasta kertovien vektorikellojen kanssa. Rinnakkaisuuden hallintakomponentti, joka on nyt osa tietovarantopalvelua, päivittää vektorikelloja kasvattamalla paikallisen solmun laskurin arvoa kunkin tallennusoperaation yhteydessä. Asiakassovelluksen hakiessa ja tallentaessa tietosisältöolioita tietovarantopalvelusta välitetään niiden mukana myös vektorikellot. Näistä lähtökohdista vektorikellojen varaan voidaan nyt rakentaa yksi yhteinen optimistisen rinnakkaisuuden hallinnan mekanismi, joka tarkkailee sekä asiakassovelluksilta saapuvien että toisil-

ta solmuilta vastaanotettujen päivitysten välisiä konflikteja. Vektorikelloa käytetään siis sekä tavanomaisen palvelukeskeisen arkkitehtuurin optimistisen lukituksen toteutusmekanismina että optimistisen replikoinnin rinnakkaisuuden hallinnan mekanismina.



Kuva 14 Hahmotelma vaihtoehtoisesta arkkitehtuurista

Tiedonsiirtokomponentti voidaan nähdä muissa solmuissa toimivien asiakassovellusten edustajana paikallisessa solmussa. Se suorittaa niiden tekemät muutokset paikallisessa solmussa vaihtelevan pituisella viiveellä riippuen siitä, miten hyvin verkkoyhteydet toimivat. Vastaavasti se suorittaa muilta solmuilta saapuvia uusien päivitysten kyselykutsuja palvelurajapintaa vasten ja välittää päivitykset paluuarvona. Rinnakkaisuuden hallintakomponentti suorittaa vektorikellopohjaisen konfliktintarkastelun kaikille päivitysoperaatioille samaa logiikkaa noudattaen, tulivat ne paikallisilta asiakassovelluksista tai etäsolmuilta. Konfliktin ratkaisu voidaan suorittaa edelleen toimialasovelluksen määrittämän konfiguraation perusteella joko automaattisesti tai antaen esimerkiksi asiakassovellusten käyttäjille mahdollisuus konfliktin ratkaisemiseen. Jos halutaan säilyttää ominaisuus, jossa konflikteja ei tarvitse ratkaista välittömästi, vaan ne voidaan säilöä tietokantaan mahdollista myöhempää ratkaisemista varten (ratkaisu voi myös tapahtua jossain muussa solmussa kuin missä konflikti alun perin tapahtui), tarvitaan tapa merkitä, mikä useasta konfliktissa olevasta tietosisältöolioversiosta on asiakassovellusten käytössä tässä solmussa. Toisin sanoen mikä olioversio on asiakassovellusten luku- ja kirjoitusoperaatioiden kohteena oleva ”työversio”, jos konfliktissa olevia olioversioita on

kannassa useita. Käytännössä yhteen olioversioista tarvitsee tehdä merkintä, joka kertoo olion olevan kyseinen solmun paikallinen ”työversio”.

Jotta tiedonsiirtokomponentti kykenee kommunikoimaan toimialasovelluksen palvelun kanssa, täytyy palvelun rajapinnan täyttää tietyt vaatimukset. Sen pitää ensinnäkin sisältää metodi, jolla voidaan kysyä tietyn ajanhetken jälkeen muuttuneet tietosisältöoliot. Näin saadaan vastattua etäsolmuilta tuleviin pollauskutsuihin. Lisäksi rajapinnassa tarvitaan tavanomaiset lisäys-, päivitys- ja poisto-operaatiot, jotta muilta solmuilta vastaanotetut tiedonkäsittelyoperaatiot saadaan vietyä tietokantaan. Siirtokomponentin vastuualueisiin kuuluu nyt myös siis hajautuspakettien deserialisointi, operaatiotyypin tarkastaminen ja tyyppiä vastaavan rajapintametodin kutsuminen. Jotta etäsolmujen tekemät pollauskutsut eivät kuormittaisi tietokantaa paikallisen solmun käyttäjien toimintaa häiriten, siirtokomponentin olisi mielekästä toteuttaa myös jonkinlainen välimuistiratkaisu, joka keventää tietokannan kuormaa.

On selvää, että tällainen arkkitehtuuri ratkaisee kahden kopion ylläpitämisen ongelman, koska tietokantoja ei ole kuin yksi, mutta järjestelmänlaajuisten invarianttien ylläpitämisen helpottuminen ei ole välttämättä itsestään selvää. Palvelukeskeisessä arkkitehtuurissa tallennettavan tiedon validointi on tyypillisesti toimialasovelluksen palvelun vastuulla, koska validointisäännöt ovat tyypillisesti toimialariippuvaisia. Nyt kun toimialasovelluksen palvelu toimii kaikkien päivitysten vastaanottajana, voi palvelu suorittaa validoinnin ja invarianttien tarkastamisen päivityksille olivat ne lähtöisin mistä tahansa. Palvelun validointitoiminnallisuutta voidaan siis uudelleen käyttää myös replikoinnin yhteydessä. Havaittaessa invariantin rikkoutuminen voitaisiin invariantin rikkova olioversio tallentaa esimerkiksi tietokantaan konfliktin muodostavana olioversiona ja suorittaa erityinen invariantin eheyttävä konfliktin ratkaisu.

5.2 Ajatuksia transaktionaalisuudesta ja relaatioista

Vaikka transaktioiden tuki rajattiinkin pois tämän työn vaatimuksista, on mielekästä tarkastella, miten niiden puuttuminen vaikuttaa replikointiratkaisun yleiseen toimivuuteen. On käytännössä äärimmäisen vaikeaa kehittää sovellusta, jonka ei tarvitsisi missään tilanteessa päivittää kahta tai useampaa tietosisältöoliota samalla kertaa yhtenä transaktionaalisena kokonaisuutena. Työssä kuvattu replikointiratkaisu käsittelee oliopäivityksiä yksittäisinä operaatioina, mikä voi aiheuttaa huomattavia ongelmia, mikäli toimialasovellus transaktioiden puuttumisesta huolimatta päivittää useampia olioita yhdellä kertaa. Ensimmäinen ongelma on, että replikointikomponentti ei millään tavalla takaa, että kaikki erilliset oliopäivitykset saapuvat tiettyyn solmuun, sillä yhteyskatkot tai tiedonsiirron pollauskutsun sopiva ajoitus voi aiheuttaa päivitysten osittaisen siirron. Myöskään mitään mekanismia, joka takaisi etäsolmulta vastaanotettujen päivitysten järjestyksen olevan sama kuin alkuperäisessä solmussa, ei ole toteutettu. Toinen ongelma on, että konfliktien tarkastelu ja ratkaisu suoritetaan aina oliokohtaisesti. Jos päivite-

tään useampaa kuin yhtä oliota, voi käydä niin, että vain yksi olioista on konfliktissa jonkin toisen päivityksen kanssa ja tähän konfliktiin tehtävä ratkaisu aiheuttaa muutoksen, joka ei enää sovi sisällöllisesti alkuperäiseen päivitysten kokonaisuuteen. Näiden ongelmien seurauksena toimialasovellus ei voi koskaan olettaa, että mikään useamman päivityksen kokonaisuus näkyisi muissa solmuissa kaikissa tapauksissa eheänä. Tämä rajoittaa toimialasovellusten logiikkaa huomattavasti tai edellyttää vähintäänkin huomattavaa valmistautumista epäeheän tiedon käsittelemiseksi.

Toinen huomattava nykyisessä ratkaisussa sivuutettu aihealue on relaatioiden ylläpitämisen ongelmat. Esimerkistä käy yksinkertainen tilanne, jossa yhteyskatkon aikana yhdessä solmussa luodaan viite yhdestä tietosisältöoliosta toiseen ja samanaikaisesti toisessa solmussa poistetaan viitattu olio. Kun yhteydet palaavat, replikointi ei havaitse konfliktia, koska viittauksen lisääminen ja viitatun olion poistaminen kohdistuivat operaatioina eri olioihin. Järjestelmässä on nyt tietosisältöolio, joka viittaa olemattomaan olioon. Käytännössä myöskään viitteitä toisiin olioihin ei siis voida käyttää toimialasovelluksissa ilman varautumista siihen, että mikä tahansa viite voi olla ”rikki”. Käytännön tasolla tämä on huomattava rajoitus toimialasovellusten logiikalle ja käy vastoin suunnitteluperiaatetta, jossa replikointilogiikka haluttiin piilottaa tietovarantopalvelurajapinnan taakse niin, että toimialasovellusten ei tarvitsisi välittää lainkaan sen olemassaolosta.

6 YHTEENVETO

Työn tavoitteena oli tutkia millainen ratkaisu voisi täyttää johtamisjärjestelmäympäristön replikoinnille asettamat vaatimukset sekä suunnitella ja toteuttaa ratkaisu osaksi palvelukeskeistä yleiskäyttöistä alustaratkaisua. Ensimmäisenä askeleena määriteltiin, mitkä johtamisjärjestelmäympäristön ominaisuudet ja erityispiirteet tulee ottaa huomioon replikointiratkaisun suunnittelussa. Seuraavaksi käytiin lävitse replikoinnin teoreettista taustaa tavoitteena muodostaa yleiskuva jo aiemmin tieteellisessä kirjallisuudessa tunnistetuista ja suunnittelussa huomioitavista asioista sekä replikoinnin eri toteutustapojen vaihtoehdoista. Tätä taustaa vasten rajattiin ne ominaisuudet, jotka johtamisjärjestelmäympäristössä toimivan replikointiratkaisun tulisi sisältää ja suunniteltiin tietovarantopalvelun taustalla toimiva ratkaisu. Erityistä huomiota kiinnitettiin vektorikellopohjaisen rinnakkaisuuden hallinnan toteutuksen yksityiskohtiin ja erityispiirteisiin. Lopuksi käytiin lävitse suunnittelu- ja toteutustyön aikana ilmenneitä arkkitehtuurin ongelmakohtia ja tehtiin mahdollisesti jatkossa hyödynnettävissä olevia parannusehdotuksia.

Työssä saatiin luotua mielekäs kokonaiskuva replikoinnin laajasta teoreettisesta kentästä ja valittua ne ominaisuudet, jotka palvelevat johtamisjärjestelmäympäristön replikoinnin toteutusta parhaiten. Optimistista replikointia hyödyntävä moni-isäntä-ratkaisu, joka noudattaa tiedon eheyden osalta ajan myötä eheä -mallia oli selkeä valinta toimintaympäristön vaatimukset huomioiden. Työssä kuvattu replikointikomponentin arkkitehtuuri ja tapa, jolla se kytkeytyy osaksi muuta alustaa, on pääpiirteissään toimiva, joskaan ei täysin ongelmaton. Tästä syystä laadittiin myös hahmotelma vaihtoehtoisesta arkkitehtuurista, joka voi ratkaista joitakin valitun arkkitehtuurin ongelmakohdista. Ehkä keskeisin optimistisen replikoinnin ongelma-alue, rinnakkaisuuden hallinta, saatiin käsiteltyä melko tyhjentävästi ja ratkaisuun saatiin suunniteltua ja toteutettua selkeä tapa konfliktien havaitsemiselle ja käsittelylle. Lisäksi tunnistettiin joitakin optimistisen replikoinnin vähemmän ilmeisiä ongelmakohtia, kuten loputon eri konfliktin ratkaisujen välisten konfliktien ratkaisukierre, ja saatiin rakennettua tapoja näiden ongelmien välttämiseksi.

Toteutettu replikointiratkaisu täyttää pääosin sille asetetut toiminnalliset vaatimukset. Näiltä osin työn voidaan katsoa onnistuneen. Ratkaisussa on kuitenkin vielä paljon ongelmakohtia, kuten järjestelmän laajuisten invarianttien ylläpitämisen haasteet ja transaktioiden tuen puuttuminen, joten suunnitellun ratkaisun ei voida vielä katsoa vastaavan kaikkiin toimintaympäristön asettamiin haasteisiin. Toisaalta, kun huomioidaan

replikoinnin ympärillä tehdyn tieteellisen tutkimustyön jo vuosikymmeniä pitkä historia ja se kuinka vähän yleispäteviä replikoinnin ratkaisumalleja näin laajan tutkimustyön pohjalta on saatu tuotettua, ei ehkä ole ihme, että tässäkään työssä ei aivan ongelmatonta ja yleiskäyttöistä ratkaisua saatu tuotettua.

LÄHTEET

[1] Pääesikunnan Materiaaliosasto. Puolustusvoimien teknologiastrategia. 2012. 9 s.

[2] Kosola, J. Verkostoliiketoiminnan malleista apua verkostopuolustusjärjestelmän luomiseen? Insinööriupseeri 2012 [verkkolehti]. s. 15-17 [viitattu 20.2.2014]. Saatavissa: http://www.iul.fi/@Bin/135130/Insin%C3%B6%C3%B6riupseeri+2012+verkkoversio.pdf

[3] Maanpuolustuskorkeakoulu. Turvallinen Suomi - Tietoja Suomen kokonaisturvallisuudesta, Verkkoversio - Laajat artikkelit. Helsinki 2013. 325 s.

[4] Peltoniemi, R. Maavoimien taistelutapa uudistettu [WWW]. [viitattu 24.2.2014]. Saatavissa: http://www.puolustusvoimat.fi/wcm/Erikoissivustot/Trombi13/Suomeksi/Maavoimien+uudistettu+taistelutapa2/

[5] Puolustusvoimien Pääesikunta/Suunnitteluosasto, Kenttäohjesääntö – Yleinen osa, Helsinki 2008.

[6] Insta DefSec Oy, Johtamisen ratkaisut - Jalkautuneen taistelijan järjestelmä [WWW]. [viitattu 26.2.2014]. Saatavissa: http://publicsafety.insta.fi/solutions-pr-fi/product-pr-fi?productid=27553168&solutionid=27547515

[7] Tanenbaum, A.S. & Van Steen, M. Distributed Systems – Principles and Paradigms. Second Edition. Upper Saddle River, NJ, USA 2007. Pearson Prentice Hall. 686 p.

[8] Saito, Y. & Shapiro, M. Replication: optimistic approaches. HP Laboratories Palo Alto 2002. 33 p. Saatavissa: http://www.hpl.hp.com/techreports/2002/HPL-2002-33.pdf?q=optimistic

[9] Özsu, M. T. & Valduriez, P. Principles of Distributed Database Systems. Third Edition. 2011. Springer Science & Business Media.

[10] Chandhok, N. Web Distribution Systems: Caching and Replication. Ohio State University 2000. Saatavissa: http://www.cse.wustl.edu/~jain/cis788-99/ftp/web_caching.pdf

[11] Backup [WWW]. [Viitattu 15.5.2014]. Saatavissa: http://en.wikipedia.org/wiki/Backup

[12] Fischer, M.J., Lynch, N.A. & Paterson, M.S. Impossibility of distributed consensus with one faulty process. Journal of the ACM (JACM) 32(1985)2, pp 374-382. Saatavissa: http://macs.citadel.edu/rudolphg/csci604/ImpossibilityofConsensus.pdf

[13] Gilbert, S. & Lynch, N.A. Perspectives on the CAP Theorem. Computer 45(2012)2, pp. 30-36. Saatavissa: http://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf

[14] Gilbert, S. & Lynch, N.A. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. ACM SIGACT News 33(2002)2, pp. 51-59.

[15] Birman K.P. Guide To Reliable Distributed Systems. London 2012. Springer. 730 p.

[16] Brewer, E. CAP twelve years later: How the" rules" have changed. Computer 45(2012)2, pp. 23-29. Saatavissa: http://www.computer.org/cms/Computer.org/ComputingNow/homepage/2012/0512/T_CO2_CAP12YearsLater.pdf

[17] Charron-Bost, B., Pedone, F., & Schiper, A. Replication - Theory and Practice. Volume 5959. Berlin 2010, Springer. 290 p.

[18] Saito, Y. & Shapiro, M. Optimistic replication. ACM Computing Surveys (CSUR) 37(2005)1, pp. 42-81. Saatavissa: http://www.ece.cmu.edu/~ece845/docs/optimistic-data-rep.pdf

[19] Types of Consistency. Colorado State University. Lecture Notes and Examples, CS 551: Distributed Operating Systems. Saatavissa: http://www.cs.colostate.edu/~cs551/CourseNotes/Consistency/TypesConsistency.html

[20] Weak consistency [WWW]. [Viitattu 10.8.2014]. Saatavissa: http://en.wikipedia.org/wiki/Weak_consistency

[21] Partially ordered set [WWW]. [Viitattu 20.9.2014]. Saatavissa: http://en.wikipedia.org/wiki/Partially_ordered_set

[22] Total order [WWW]. [Viitattu 22.9.2014]. Saatavissa: http://en.wikipedia.org/wiki/Total_order

[23] Mills, D. Network Time Protocol (Version 3) specification, implementation and analysis, RFC1305. 1992. Saatavissa: http://tools.ietf.org/html/rfc1305

[24] Cristian, F. Probabilistic clock synchronization. Distributed computing 3(1989)3, pp. 146-158. Saatavissa: http://kom.aau.dk/~rlo/lectures/algoAndArchIII_2009/ChristiansAlgorithm.pdf

[25] Gusella, R. & Zatti, S. The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD. IEEE Transactions on Software Engineering 15(1989)7, pp. 847-853.

[26] Dana, P.H. Global Positioning System (GPS) time dissemination for real-time applications. Real-Time Systems 12(1997)1, pp. 9-40. Saatavissa: http://pdana.com/PHDWWW_files/Rtgps.pdf

[27] Lamport, L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(1978)7, pp. 558-565.

[28] Fidge, C.J. Timestamps in message-passing systems that preserve the partial ordering. Proceedings of the 11th Australian Computer Science Conference. 1988. pp. 56-66. Saatavissa: http://zoo.cs.yale.edu/classes/cs426/2012/lab/bib/fidge88timestamps.pdf

[29] Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., & Alonso, G. Understanding replication in databases and distributed systems. Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS'2000), Taiwan, Taipei, April 2000. IEEE. pp. 264–274. Saatavissa: http://faculty.ksu.edu.sa/metwally/IS%20511/understanding%20replication.pdf

[30] Defago, X., Schiper, A., & Sergent, N. Semi-passive replication. Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS), West Lafayette, IN, USA, October 1998. IEEE. pp. 43–50. Saatavissa: http://www.jaist.ac.jp/~defago/files/pdf/DSS98.pdf

[31] High-Availability Database (HA DB) [WWW]. [Viitattu 25.6.2014]. Saatavissa: http://raima.com/rdme-high-availability-database/

[32] Gowans, D. Asynchronous or Synchronous Replication..? [WWW]. [Viitattu 15.7 2014]. Saatavissa: http://blogs.msdn.com/b/douggowans/archive/2008/12/12/asynchronous-or-synchronous-replication.aspx

[33] Replication [WWW]. [Viitattu 15.8.2014]. Saatavissa: http://redis.io/topics/replication

[34] Skeen, D & Stonebraker, M. A formal model of crash recovery in a distributed system. IEEE Transactions on Software Engineering 9(1983)3, pp. 219-228. Saatavissa: http://www.inf.fu-berlin.de/lehre/SS10/DBS-TA/Reader/3PCSkeenStonebr.pdf

[35] Aguilera, M. K., Merchant, A., Shah, M., Veitch, A. & Karamanolis, C. Sinfonia: a new paradigm for building scalable distributed systems. SOSP '07: Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles. New York, NY, USA 2007, ACM. pp. 159-174. Saatavissa: http://www.cs.princeton.edu/courses/archive/fall08/cos597B/papers/sinfonia.pdf

[36] Gray, J. & Lamport, L. Consensus on transaction commit. ACM Transactions on Database Systems 31(2006)1, pp. 133-160. Saatavissa: http://www.inf.fu-berlin.de/lehre/SS10/DBS-TA/Reader/transactionCommitgray-06.pdf

[37] Lamport, L. Paxos made simple. ACM Sigact News, 32(2001)4, pp. 18-25. Saatavissa: http://www.cs.utexas.edu/users/lorenzo/corsi/cs380d/past/03F/notes/paxos-simple.pdf

[38] Chandra, T. D., Griesemer, R. & Redstone, J. Paxos made live: an engineering perspective. Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing. Portland, OR 2007, ACM. pp. 398-407. Saatavissa: http://www.cs.cmu.edu/~pavlo/courses/fall2013/static/papers/p398-chandra.pdf

[39] Ongaro, D., & Ousterhout, J. In search of an understandable consensus algorithm. Proceedings of the 2014 USENIX Annual Technical Conference 2014. Philadelphia, PA June 2014. Saatavissa: http://ramcloud.stanford.edu/raft.pdf

[40] Parker Jr, D. S., et al. Detection of mutual inconsistency in distributed systems. IEEE Transactions on Software Engineering 9(1983)3, pp. 240-247.

Saatavissa:
http://zoo.cs.yale.edu/classes/cs426/2012/bib/parker83detection.pdf

[41] Brewer, Eric. CAP twelve years later: How the" rules" have changed. Computer 45(2012)2, pp. 23-29. Saatavissa:
http://www.realtechsupport.org/UB/NP/Numeracy_CAP%2B12Years_2012.pdf

[42] Letia, M., Preguiça, N. & Shapiro, M. CRDTs: Consistency without concurrency control. Rocquencourt, France 2009, Institut National de la Recherche en Informatique et Automatique (INRIA), Rapport de recherche n°6956. 13 p. Saatavissa: http://arxiv.org/pdf/0907.0929

[43] Alvaro, P., Conway, N., Hellerstein, J. M. & Marczak, W. R. Consistency Analysis in Bloom: a CALM and Collected Approach. Proceedings of the 5th Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA 2011. pp. 249-260.

[44] JSR 338. Java Persistence API, Version 2.1. 2013, Oracle. 570 p. Saatavissa:
http://download.oracle.com/otndocs/jcp/persistence-2_1-fr-eval-spec/index.html

[45] Hibernate User Guide, Timestamp [WWW]. [Viitattu 2.2.2015]. Saatavissa:
http://docs.jboss.org/hibernate/orm/5.0/userGuide/en-US/html_single/#d5e1240

[46] Gossip protocol [WWW]. [Viitattu 13.2.2015]. Saatavissa:
http://en.wikipedia.org/wiki/Gossip_protocol